

E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Sistema IoT para la monitorización y visualización de parámetros ambientales en entornos industriales



Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Iñaki Ibiricu Elizondo

Santiago Tainta Ausejo

Pamplona, 11 de junio de 2020

RESUMEN

Este trabajo final de grado ha sido realizado en la empresa Tasubinsa, que propuso la idea de desarrollar un sistema para monitorizar los parámetros de ciertos procesos productivos de la fábrica. El objetivo del proyecto es implementar una red IoT (Internet of Things) en diferentes líneas de producción usando motas repartidas por la fábrica. Las motas empleadas están basadas en el microcontrolador ESP32 y captan datos ambientales a través de varios sensores que enviarán a través de Wifi. Estos datos son procesados por un broker MQTT y enviados para su almacenamiento a una base de datos a través de una aplicación llamada Telegraf. Para acceder a la información se usará una aplicación web donde se podrá representar en gráficas y tablas con la finalidad de visualizarla de una manera más fácil y agradable.

Finalmente, el proyecto cuenta con un servidor DHCP para poder asignar direcciones IP a todos los dispositivos de la red IoT y una página web de gestión. Esta página web está securizada con certificados SSL y autenticación de usuario, permitiendo la interacción con todas las aplicaciones usadas.

LISTA DE PALABRAS CLAVE

- Internet de las cosas
- Red de sensores
- Docker
- MQTT
- ESPHome

SUMMARY

This final degree work has been carried out in the company Tasubinsa, which proposed the idea of developing a system to monitor the parameters of certain production processes in the factory. The aim of the project is to implement an IoT (Internet of Things) network on different production lines using motes spread around the factory. The motes used are based on the ESP32 microcontroller and capture environmental data through various sensors, sending the information via Wifi. The data is processed by a MQTT broker and sent to a database for storage through an application called Telegraf. To access the information a web application is used where it can be represented in graphs and tables in order to visualize it in an easier and more pleasant way.

Finally, the project has a DHCP server to assign IP addresses to all the devices of the IoT network and a management web page. This web page is secured with SSL certificates and user authentication, allowing interaction with all the applications used.

KEYWORDS

- Internet of Things
- Sensor network
- Docker
- MQTT
- ESPHome

ÍNDICE

1.	Introducción	1
1.1	Introducción	1
1.2	Objetivo	1
1.3	Metodología	2
2.	Estado del Arte	4
2.1	Introducción al Internet of Things	4
2.2	Comunicación	5
2.3	Estructura de un sistema IoT	6
3.	Arquitectura del sistema	8
3.1	Funcionamiento	8
3.2	Esquema general	9
4.	Elementos Hardware	11
4.1	Dispositivos ESP8266 y ESP32	11
4.2	Variables a monitorizar y sensores utilizados	13
4.2.1	Temperatura y Humedad	13
4.2.2	CO ₂ y COVs	14
4.2.3	CO y Formaldehído	15
4.2.4	Temperatura del agua	20
4.2.5	Conductividad	22
4.2.6	Control de pH	24
4.3	Conexiones	26
4.3.1	Pin GPIO	26
4.3.2	Bus I ² C	27
4.3.3	Señales Analógicas	28
4.4	Esquema	31
5.	Software	33
5.1	Instalación del proyecto	33
5.1.1	Máquina Virtual	33
5.1.2	Dockers	34
5.2	Programación de los dispositivos	35
5.2.1	Servidor DHCP	35
5.2.2	ESPHome	36
5.2.3	Configuración de dispositivos	36
5.2.4	Programación de los sensores	38
5.2.5	ESPHome-Flasher	39
5.2.6	Actualización vía OTA (Over The Air)	39
5.3	Monitorización y almacenamientos de datos	40
5.3.1	MQTT	40
5.3.2	InfluxDB	42
5.3.3	Telegraf	44
5.4	Administración y visualización de datos	45
5.4.1	Grafana	45
5.4.2	Portainer	46
5.4.3	Nginx	47
6.	Problemas	50
7.	Conclusiones y líneas futuras	51
8.	Bibliografía	52

A.	Código del proyecto en Linux (Ubuntu)	54
A.1	Instalar Docker en Ubuntu	54
A.2	Instalar Portainer.io en un contenedor	54
A.3	Instalar MQTT en un contenedor.....	55
A.4	Instalar ESPHome en un contenedor	56
A.5	Instalar InfluxDB en un contenedor	56
A.6	Instalar Telegraf en un contenedor	58
A.7	Instalar DHCP en un contenedor	59
A.8	Instalar Grafana en un contenedor.....	60
A.9	Instalar Nginx en un contenedor	60
B.	Programación del dispositivo ESP32	64

ÍNDICE DE FIGURAS

Figura 1:[Internet of Things].....	4
Figura 2:[Estructura IoT].....	6
Figura 3:[Diagrama de conexión].....	7
Figura 4:[Esquema del flujo de información].....	9
Figura 5:[Esquema máquina virtual 1]	10
Figura 6:[Esquema máquina virtual 2]	10
Figura 7:[Conexiones ESP01]	11
Figura 8:[Dispositivo ESP32].....	12
Figura 9:[Conexiones ESP32]	12
Figura 10:[Sensor DHT11].....	13
Figura 11:[Trama de datos sensor DHT11].....	13
Figura 12:[Características sensor DHT11]	14
Figura 13:[Sensor DHT22].....	14
Figura 14:[Sensor CCS811].....	15
Figura 15:[Circuito interno sensor MQ]	16
Figura 16:[Esquema de pines sensor MQ]	16
Figura 17:[Sensor MQ7].....	17
Figura 18:[Curva logarítmica de sensibilidad sensor MQ7].....	17
Figura 19:[Curva de sensibilidad sensor MQ7]	18
Figura 20:[Sensor MQ138]	19
Figura 21:[Curva logarítmica sensibilidad e influencia de la temperatura sensor MQ138]	19
Figura 22:[Curva sensibilidad sensor MQ138]	19
Figura 23:[Sensor DS18B20]	20
Figura 24:[Error de desviación sensor DS18B20].....	21
Figura 25:[Tabla de resoluciones del DS18B20].....	21
Figura 26:[Características sensor DS18B20].....	22
Figura 27:[Sensor DFR0300]	22
Figura 28:[Sistema acondicionamiento sensor DFR0300]	23
Figura 29:[Características sensor DFR0300].....	24
Figura 30:[Sensor de pH SEN01061]	24
Figura 31:[Relación voltaje y valor de pH].....	25
Figura 32:[Relación de conversión de voltaje a pH]	25
Figura 33:[Sistema acondicionamiento sensor SEN0161]	25
Figura 34:[Características sensor SEN0161].....	26
Figura 35:[Montaje sensor DHT11]	27
Figura 36:[Montaje sensor DHT22]	27
Figura 37:[Montaje sensor DS18B20].....	27
Figura 38:[Montaje sensor CSS811].....	28
Figura 40:[Comportamiento sensor analógico].....	29
Figura 39:[Esquema sensor analógico].....	29
Figura 41:[Montaje sensor MQ7]	29
Figura 42:[Montaje sensor MQ138]	30
Figura 43:[Montaje sensor DFR0300].....	30
Figura 44:[Montaje sensor SEN0161]	31
Figura 45:[Montaje completo proyecto]	31
Figura 46:[Montaje real del proyecto].....	32
Figura 47:[Configuración NAT]	33
Figura 48:[Configuración BRIDGE].....	33
Figura 49:[Esquema estructura Docker]	34
Figura 50:[Interfaz web servidor DHCP]	35

Figura 51:[Interfaz web ESPHome].....	37
Figura 52:[Configuración de un nodo en ESPHome].....	37
Figura 53:[Nodo ESP32].....	38
Figura 54:[Sensores disponibles en ESPHome]	38
Figura 55:[Flasheo de dispositivo ESP32]	39
Figura 56:[Actualización vía OTA]	40
Figura 57:[Funcionamiento bróker MQTT].....	41
Figura 58:[Mensaje entrantes al bróker MQTT]	41
Figura 59:[Tabla de datos de InfluxDB]	43
Figura 60:[Política de retención de InfluxDB]	43
Figura 61:[Usuarios de InfluxDB].....	43
Figura 62:[Panel gráfico de Grafana]	46
Figura 63:[Interfaz web Portainer]	46
Figura 64:[Página web]	48
Figura 65:[Autenticación de usuario]	49

1. Introducción

1.1 Introducción

La empresa Talleres Auxiliares de Subcontratación Industrial Navarra, Sociedad Anónima, más comúnmente conocida como Tasubinsa, es un Centro Especial de Empleo, CEE, que ofrece recursos de subcontratación industrial y de servicios a sus empresas clientes, proporcionados por una plantilla compuesta en su mayor parte por personas con discapacidad intelectual. Esta empresa se centra en tres grandes líneas de negocio: subcontratación industrial, inyección de plástico y división de servicios.

- **Subcontratación industrial:** especializados en el montaje de subconjuntos o conjuntos finales de todo tipo, integrando la inyección de las piezas y su decoración, cableados eléctricos, compra de materiales, o sirviendo los productos en secuenciación en la cadena de montaje de las empresas clientes. Los diferentes centros de trabajo de Tasubinsa están dotados con equipamientos de referencia para satisfacer las necesidades de subcontratación con tecnología avanzada y personal cualificado.
- **División de Servicios:** se realizan trabajos de jardinería y forestales, recogida de basuras, compostaje, limpieza viaria e industrial, de oficinas, finales de obra y organización de eventos. Las brigadas de Tasubinsa están dotadas con una amplia variedad de maquinaria para realizar los más diversos servicios y cuentan con personal multidisciplinar cualificado para satisfacer sus necesidades.
- **Inyección de plástico:** es una de las líneas industriales más relevantes y diferenciadoras de Tasubinsa. Cuenta con 34 máquinas de inyección y se fabrican anualmente más de 142 millones de piezas de plástico que posteriormente se ensamblan con otros componentes, en la mayoría de los casos, ofreciendo un servicio integral y unificado.

Uno de los planes del departamento de industrialización de la empresa Tasubinsa es sensorizar las plantas para tener información de parámetros importantes sobre el estado de algunos procesos y evaluar determinadas situaciones para poder actuar de manera rápida y eficaz. Para ello, se planteó la idea de crear una red de IoT (Internet of Things) con varios dispositivos ESP32 repartidos por los distintos puntos de la planta para captar valores de temperatura, humedad y gases contaminantes. De esta manera se podría monitorizar estos datos en los diferentes puestos de trabajo donde se utilizan componentes químicos, normalmente contaminantes para la salud humana, y así poder enviar avisos si los niveles superasen unos límites para poder proteger la salud de los operarios.

1.2 Objetivo

Este proyecto se va a realizar en la empresa Tasubinsa y consiste en crear una red IoT (Internet of Things) con distintos dispositivos ESP32 repartidos por la fábrica que captan datos de varios sensores. Estos datos son enviados mediante MQTT a una base de datos a través de un programa llamado Telegraf. También se utilizará un programa llamado Grafana que se conecta a la base de datos para poder representar los datos en gráficas y tablas.

Mediante una página web segura con certificados SSL y autenticación de usuario se podrá acceder a todos los programas e interactuar con ellos. Finalmente, el proyecto cuenta con un servidor DHCP para poder asignar direcciones IP a todos los dispositivos de la red IoT.

1.3 Metodología

A la hora de abordar el proyecto de sensorizar las plantas, Tasubinsa valoró las opciones existentes en el mercado y tras un periodo de obtener información acerca del IoT se tuvieron en cuenta dos alternativas:

1. Buscar en el mercado dispositivos que cumpliesen en la medida de lo posible las expectativas del proyecto y que incorporasen una amplia gama de sensores para poder captar los máximos parámetros posibles. El problema de esta alternativa es que estos dispositivos enviaban los datos a un servidor propio de la marca del dispositivo y la empresa no tenía control sobre estos, es decir, solo podían visualizarlos en una interfaz propia de la marca.
2. Desarrollar un sistema propio que incluyera todas las aplicaciones necesarias para la captura, almacenamiento y visualización de datos. Esta idea requeriría montar todo desde cero, empezando por la programación de los sensores y las motas hasta la instalación del software completo en el servidor de la empresa. El problema de esta alternativa era que los conocimientos sobre el tema eran escasos y habría que invertir bastante tiempo en hacer pruebas y desarrollar un primer sistema eficiente.

Tras valorar ambas opciones, la empresa optó por desarrollar una red IoT propia, ya que así tendrían un control total sobre el flujo de datos, el coste a largo plazo sería menor y se tendría la capacidad de ampliar el sistema y mejorarlo en un futuro, al no depender de empresas externas. Para empezar, la empresa desarrolló un sistema sencillo con dispositivos ESP01 y únicamente con sensores de temperatura. Este sistema recopilaba la temperatura de todas las oficinas y mandaba los datos a un bróker a través de MQTT. Debido a la sencillez, no se cumplían los requisitos necesarios para implementarlo en un proceso productivo. Se encontraron los siguientes inconvenientes:

- El sistema solo mandaba los datos a un servidor MQTT. Esto no resultaba práctico ya que este servidor solo captaba los datos de los sensores, pero no los almacenaba en ninguna base de datos.
- No existía ninguna aplicación para poder representar los datos de manera más visible y práctica para los usuarios que iban a usar el sistema. En este caso, sería conveniente disponer de un panel de mandos con gráficas y tablas de los datos captados en tiempo real.
- Los dispositivos solo captaban datos de sensores de temperatura, lo que supone poca información en caso de implantarse en un proceso productivo. Eran necesarios más datos como humedad, CO₂, pH, etc.
- El programa utilizado para programar el firmware era bastante simple y solo funcionaba bien en dispositivos con pocos sensores y actuadores.

Debido a esta situación, se decidió utilizar unos dispositivos más complejos que permitiesen la conexión de más dispositivos para así poder ampliar la red IoT y representar los datos en una aplicación web. Mi labor en Tasubinsa ha consistido en desarrollar una red IoT empleando varios nodos repartidos por la fábrica para la captación de datos en los procesos de fabricación como la inyección de plásticos o en la calidad de las aguas residuales entre otros usos.

Inicialmente fue necesario un periodo de adaptación y formación antes de empezar a realizar tareas del proyecto. Durante este periodo se llevó a cabo una labor de investigación para adquirir los conocimientos necesarios para su realización, ya que estos no habían sido cubiertos durante los estudios de grado, así como una primera toma de contacto con el sistema existente.

Posteriormente, se buscó información sobre todas las aplicaciones existentes en el mercado, especialmente aquellas de código abierto, y se decidió desarrollar un sistema en una máquina virtual donde todas las aplicaciones estuviesen alojadas en contenedores. Se decidió emplear comunicación vía Wifi para poder programar dispositivos con sensores específicos para cada zona de la fábrica. Esos datos serán enviados al servidor MQTT y almacenados en una base de datos para poder hacer históricos. Para la visualización de estos se utilizará una aplicación web responsable de representar los datos gráficamente.

Como se puede observar, inicialmente el proyecto era muy ambicioso y de gran envergadura. Sin embargo, durante el desarrollo del mismo ha habido un grave problema (el coronavirus) que me ha impedido acceder a los recursos que me ofrece la empresa, por lo que existen algunos aspectos del proyecto que no se han podido realizar. Las principales tareas que no se han podido desarrollar en su totalidad han sido la programación de todos los sensores debido a que no estuvieron disponibles hasta finales de mayo y la configuración del servidor DHCP debido a que ha sido imposible volcar el proyecto en un servidor de la empresa por motivos de seguridad.

2. Estado del Arte

2.1 Introducción al Internet of Things

Podemos definir IoT (Internet of Things) como la agrupación e interconexión de dispositivos y objetos a través de una red (bien sea privada o Internet), dónde todos ellos podrían ser visibles e interaccionar. Podemos conectar objetos o dispositivos de cualquier tipo, desde sensores y dispositivos mecánicos hasta objetos cotidianos como pueden ser el frigorífico, el calzado o la ropa. Cualquier cosa que se pueda imaginar podría ser conectada a internet e interaccionar sin necesidad de la intervención humana, resultando una interacción máquina a máquina (M2M) [1].

Se puede considerar que el primer dispositivo IoT apareció en el año 1990 cuando John Romkey y Simon Hacket consiguieron diseñar una tostadora con conectividad a Internet, pudiendo desde cualquier ordenador determinar su encendido, su apagado y configurar el tiempo de tostado de esta. A partir de este momento, se supo el potencial que tendría la posibilidad de conectar ciertos dispositivos a Internet, pero no fue hasta 1999 cuando el ingeniero Bill Joy, se percató de la importancia que tendría este aspecto a la hora de automatizar y disponer de control sobre una multitud impensable de dispositivos.

Internet ha evolucionado rápidamente y esto ha permitido que el Internet of Things sea ya una realidad en nuestros días y no una visión de futuro. La fama de esta tecnología radica principalmente en todas las aplicaciones y posibilidades que nos proporciona tanto para mejorar la vida cotidiana de las personas como los entornos empresariales.



Figura 1:[Internet of Things]

Sin embargo, hay varios obstáculos que están retrasando el rápido desarrollo del IoT. De todos ellos, los tres más importantes son la implementación de IPv6, la energía de los sensores y un acuerdo sobre los estándares.

- **La implementación de IPv6.** El mundo se quedó sin direcciones IPv4 en febrero de 2010. Si bien el público general no se ha visto afectado por ningún impacto real, esta situación podría frenar potencialmente el avance del IoT, ya que los miles de millones de nuevos sensores potenciales necesitarán direcciones IP únicas. Además, IPv6 facilita la gestión de las redes gracias a las capacidades de configuración automática y también ofrece características de seguridad mejoradas.

- **Energía de los sensores.** Para que el IoT alcance su pleno potencial, los sensores deberán ser autosuficientes ya que no es viable tener que cambiar las baterías de miles de millones de dispositivos implementados por todo el planeta, e incluso en el espacio. Por tanto, lo que se necesita es una forma de que los sensores generen electricidad a partir de elementos del medio ambiente, como las vibraciones, la luz y el flujo de aire.

Esta área de trabajo se conoce como Energy Harvesting y es un campo de investigación muy activo en la actualidad. Por ejemplo, en un avance significativo, se ha presentado un nanogenerador [2] viable comercialmente en la Reunión y Exposición Nacional de la Sociedad Estadounidense de Química, celebrada en marzo de 2011. Este dispositivo se trata de un chip flexible que utiliza los movimientos del cuerpo, como la pulsación con el dedo, para generar electricidad.

- **Estándares.** Si bien se ha avanzado mucho en el ámbito de los estándares, es necesario realizar más avances, especialmente en lo que respecta a seguridad, privacidad, arquitectura y comunicaciones. El Instituto de Ingeniería Eléctrica y Electrónica (IEEE) [3] es solo una de las múltiples organizaciones que trabajan para solucionar estos problemas, asegurándose de que los paquetes IPv6 puedan enviarse a través de diferentes tipos de redes.

Es importante señalar que, si bien existen barreras y retos, estos no son insalvables. Dadas las ventajas del IoT, estas cuestiones se están solucionando progresivamente con el paso del tiempo.

2.2 Comunicación

En este proceso de comunicación es donde IoT está evolucionando más rápidamente ya que uno de los principales escollos a salvar es el tipo de protocolo con el que se comunican los dispositivos (es decir, "el idioma" que hablan entre ellos). Actualmente, existen dispositivos o sensores muy nuevos cuya comunicación y conexión a internet es fácil y directa, pero también existen muchos otros dispositivos más antiguos no estándar cuyo protocolo de comunicación y conexión no es trivial.

Uno de los mecanismos que se ha intentado establecer es un protocolo abierto denominado MQTT [4] (Message Queuing Telemetry Transport) facilitando así la comunicación entre distintos dispositivos de diferentes fabricantes. Más adelante, se explicará más detalladamente como funciona este protocolo y cuál es su función en este proyecto.

Otro aspecto tecnológicamente importante para habilitar el IoT es la tecnología utilizada para la comunicación entre varios dispositivos cuya ubicación no sea próxima, es decir, las redes de comunicación inalámbricas. Es posible dividir las mismas en 3 grandes bloques:

- **Las tecnologías tradicionales de conectividad inalámbrica:** el Wifi y la conectividad celular (del 2G al 4G), ambas de alto consumo energético, pero ampliamente soportadas y con gran cobertura.
- **Las tecnologías de corto alcance:** han sido claves para el éxito inicial del IoT, pero en muchos casos son poco eficaces en despliegues amplios. Es posible hablar de tecnologías como ZigBee, Z-Wave, 6LoWPAN, etc.

- **Nuevas tecnologías nativas de comunicación:** son tecnologías de muy bajo consumo, largo alcance y con bajo coste de dispositivos, de las que podemos destacar Sigfox y Lora.

2.3 Estructura de un sistema IoT

El mercado IoT ofrece cada vez más opciones de plataformas, soluciones y aplicaciones en la nube. Algunas son más o menos cerradas y otras permiten la construcción de la plataforma IoT desde la base.

Para las aplicaciones de IoT basadas en la captación de datos y en el análisis y visualización de estos es muy habitual el uso de la arquitectura de tres niveles (*Figura 2*). En esta arquitectura se captura el flujo de información desde los dispositivos los cuales lo transfieren a diversos servicios ubicados en la nube. Las tres capas que la componen son las siguientes:

- **Capa de dispositivos:** es la capa inferior de la arquitectura. Hay varios tipos de dispositivos, pero para que se consideren dispositivos IoT deben tener algún tipo de comunicación, directa o indirecta, que lo enlaza con Internet.
- **Capa de comunicación:** esta capa soporta la conectividad de los dispositivos. Hay múltiples protocolos para la comunicación entre los dispositivos y la nube entre los que destacan HTTP, MQTT y CoAP.
- **Capa de agregación:** es una capa importante de la arquitectura ya que hace de bróker de comunicaciones, ofreciendo soporte para un servidor MQTT para hablar con los dispositivos, enrutamiento de comunicaciones a dispositivos específicos y la habilidad de hacer un puente y transformar diferentes protocolos.

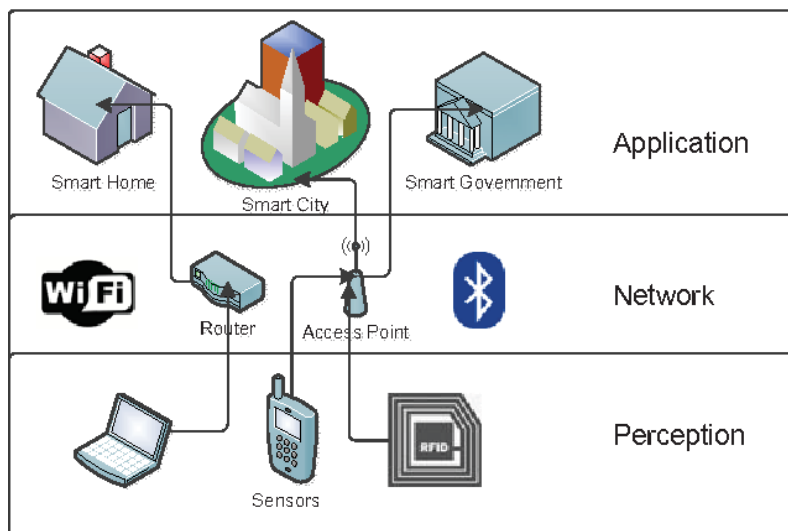


Figura 2:[Estructura IoT]

Además de estas tres capas básicas, una arquitectura de IoT más completa también incluiría capas que ayudan a la gestión de identidades o la seguridad de los datos.

La integración de un sistema IoT a cualquier proceso productivo requiere tres fases: conexión, análisis y visualización y automatización.

Fase 1: Conexión. Los dispositivos conectados a una red con sensores inteligentes empiezan a enviar información sobre sí mismos y su entorno a un centro de comunicaciones en la nube. Conectar cosas (darles sentido y abrirles una conexión a Internet para que puedan enviar sus datos) representa el comienzo de la evolución del IoT.

Fase 2: Análisis y Visualización. A medida que los datos se acumulan hay que empezar a ejecutar análisis inteligentes de las pilas de datos y visualizar los resultados en los dashboards.

Fase 3: Automatización. Con una visión clara de todos los datos que se recopilan ya se puede pensar en hacer automatizaciones para aprovechar todo el potencial del IoT. Entre las más usadas se encuentran la configuración de alarmas, tareas y análisis.

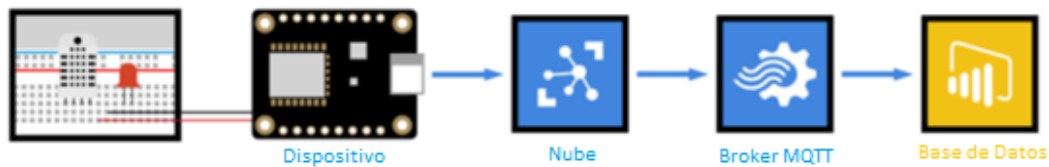


Figura 3:[Diagrama de conexión]

3. Arquitectura del sistema

3.1 Funcionamiento

La idea de montar una red IoT desde cero ha supuesto valorar numerosas opciones existentes en el mercado para buscar la mejor opción posible. Los conocimientos y los recursos iniciales para desarrollar un sistema propio con todos los programas necesarios para la captura, almacenamiento y visualización de datos eran escasos por eso se ha buscado una forma eficiente y barata de construir el sistema.

La primera decisión fue utilizar una comunicación a través de una red Wifi, puesto que admite una tasa de transferencia alta y el rango de alcance es suficiente para el tamaño de la red IoT. Además, no va a suponer coste alguno para la empresa ya que es una red inalámbrica muy utilizada en todas las empresas y domicilios.

Posteriormente se decidió dividir el proyecto en dos partes. Por un lado, están los elementos de Hardware, que es la parte física y se encarga de recopilar los datos del entorno que les rodea y, por el otro lado, está el Software, cuya misión es monitorizar, almacenar y visualizar los datos en un lenguaje que entendamos.

- Elementos de Hardware

Está formado por un conjunto de sensores y dispositivos cuya misión es recopilar información, adaptarla y enviarla a un bróker mediante MQTT. Los dispositivos que se usarán en el proyecto estarán basados en el microcontrolador ESP32, ya que son compactos, baratos y disponen de bastantes pines de entrada/salida, lo que va a permitir conectar varios sensores a un único dispositivo.

Se usarán sensores de pequeño tamaño capaces de detectar determinados gases, la temperatura y la humedad con una precisión bastante fiable. También se utilizarán otros sensores específicos para el tratamiento de aguas capaces de medir el pH, la conductividad y la temperatura.

- Software

Esta parte está formada por un conjunto de aplicaciones instaladas en contenedores cuya misión es recopilar, almacenar y visualizar los datos enviados por los dispositivos ESP32. Estos datos van a parar a un bróker que se encarga de recopilar todos los datos que se envían a través del protocolo MQTT. Otro programa llamado Telegraf se encargará de subscribirse a este bróker para captar estos datos y almacenarlos en una base de datos. Para visualizar los datos se utilizará un programa llamado Grafana que leerá los datos de la base de datos y los representará en diferentes gráficas y tablas. El flujo de la información en el sistema se puede ver en la *Figura 4*.



Figura 4:[Esquema del flujo de información]

Por otro lado, se instalará un contenedor con ESPHome para poder gestionar los dispositivos ESP32 de forma remota y actualizarlo vía OTA (Over The Air) y otro contenedor con un servidor DHCP para poder asignar direcciones IP únicas a todos los dispositivos ESP32 conectados a la red IoT. Además, se creará un portal web totalmente seguro para poder administrar todos los contenedores e interactuar con ellos.

Todos estos contenedores van a estar instalados en dos máquinas virtuales para que el sistema vaya fluido y el consumo de recursos no sea excesivo. El objetivo de crear dos máquinas virtuales es que una de ellas sea para gestionar el flujo de información con el exterior y la otra se destine únicamente al almacenamiento de datos.

3.2 Esquema general

Como ya hemos comentado antes, el sistema está pensado para estar instalado en dos máquinas virtuales dentro de un servidor. Cada máquina virtual tendrá su propia dirección IP y todos los programas instalados estarán dentro de contenedores ya que al ser portátiles no suponen un incremento significativo en el consumo de recursos y, además, son rápidos y fáciles de implementar. La *Figura 5* y *Figura 6* muestran un esquema de los programas que tendrá cada máquina virtual así como el flujo de información entre los contenedores y el exterior.

Como se puede observar en la *Figura 5*, esta máquina virtual tendrá instaladas seis aplicaciones que se encargarán de captar y representar los datos. El contenedor con el servidor MQTT servirá como nodo de comunicación, ya que captará los datos que envían los dispositivos y los enviará a Telegraf a través del protocolo MQTT (flecha verde). Por otra parte, el contenedor con Grafana leerá los datos de una base de datos, InfluxDB y los representará gráficamente a través de una aplicación web (flecha azul). El resto de los contenedores tienen funciones que no influyen en el flujo de los datos. El contenedor con ESPHome se encargará de dar soporte al firmware de los dispositivos, el de Nginx alojará una página web para poder interactuar con las aplicaciones del proyecto y finalmente el contenedor con Portainer se encargará de administrar y configurar todos los contenedores (flecha roja).

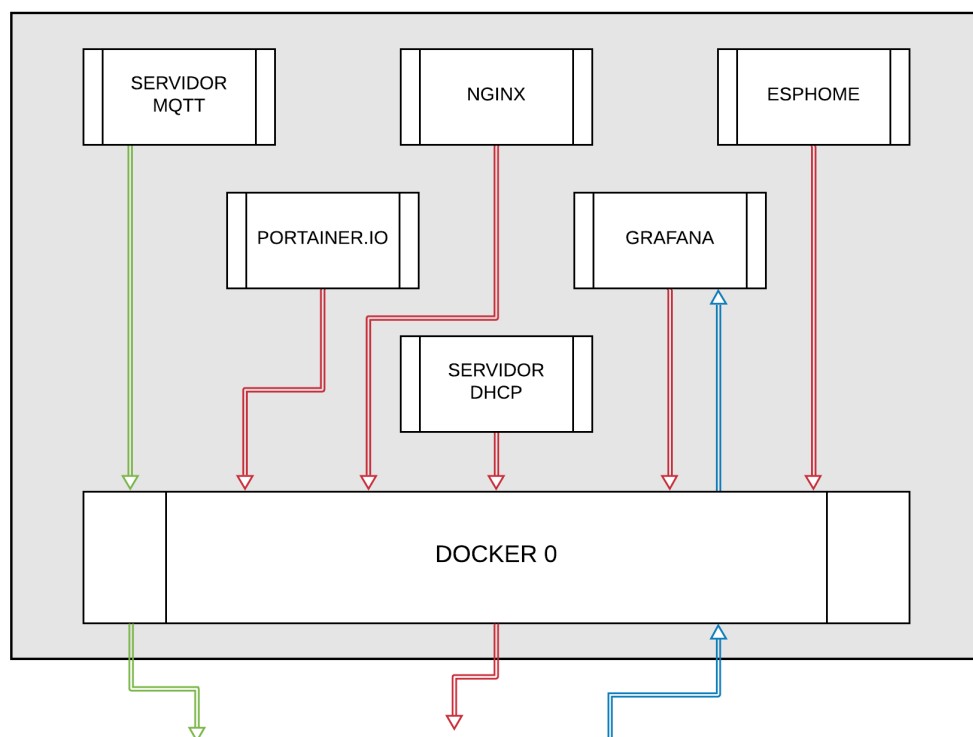


Figura 5:[Esquema máquina virtual 1]

La máquina virtual que se muestra en la *Figura 6*, se encargará principalmente de almacenar los datos. Estará formada únicamente por dos contenedores, Telegraf e InfluxDB. Su misión será capturar la información del servidor MQTT, almacenarla en la base de datos y permitir el acceso a la misma a otras aplicaciones, como por ejemplo Grafana.

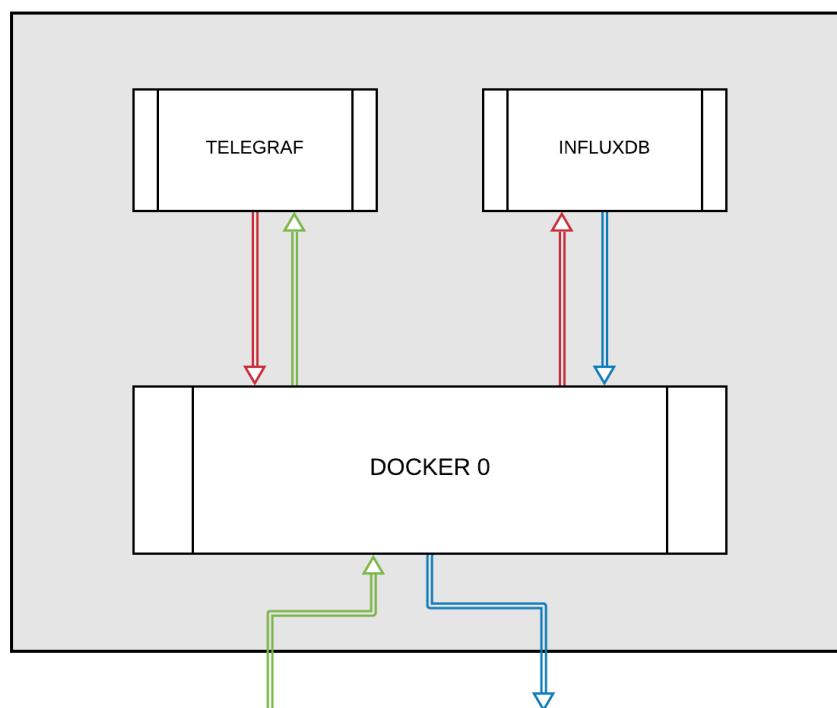


Figura 6:[Esquema máquina virtual 2]

4. Elementos Hardware

4.1 Dispositivos ESP8266 y ESP32

El ESP8266 [5] es un microcontrolador de bajo coste que apareció en el mercado a mediados del año 2014 y ha significado una pequeña revolución en el mundo del Internet of Things debido a sus capacidades de procesamiento y comunicación. Esto ha dado lugar a la existencia de múltiples placas y módulos que lo utilizan. Además, tiene la capacidad de comunicarse vía Wifi y posee una pila TCP/IP completa lo que hace que su comunicación con otros equipos o dispositivos sea bastante versátil y rápida.

Uno de los módulos que utiliza el chip ESP8266 es el ESP01 [6]. Este es el módulo más sencillo existente en el mercado, con bajo consumo energético y bajo coste. Una de sus grandes limitaciones es que posee únicamente 8 pines de conexión, los cuales debemos conocer bien para usarlos adecuadamente. Como se puede ver en la *Figura 7*, el dispositivo ESP01 posee dos pines de propósito general (GPIO) y otros dos para la transmisión de datos serie.

- **GND:** Terminal de tierra.
- **GPIO2:** Salida/Entrada de propósito general, es el pin digital 2.
- **GPIO0:** Entrada/Salida de propósito general, es el pin digital 0.
- **RXD:** Puerto de recepción de datos serie. Trabaja a 3.3V y se puede configurar como un pin digital, que en este caso sería el número 3.
- **TXD:** Puerto de transmisión de datos serie. Trabaja a 3.3V y también se puede configurar como un pin digital, en este caso el número 1.
- **CH_PD:** Este terminal se utiliza para apagar y encender el módulo ESP01. Al estar a 0V (LOW) el módulo se apaga, y a 3.3V (HIGH) se enciende.
- **RESET:** Este terminal sirve para reiniciar el módulo, si se coloca a 0V se reinicia.
- **Vcc:** Terminal para alimentar el módulo. Se debe suministrar 3.3V, pero puede soportar hasta 3.6V. Para que la conectividad Wifi funcione correctamente, la corriente suministrada debe ser superior a 200mA.

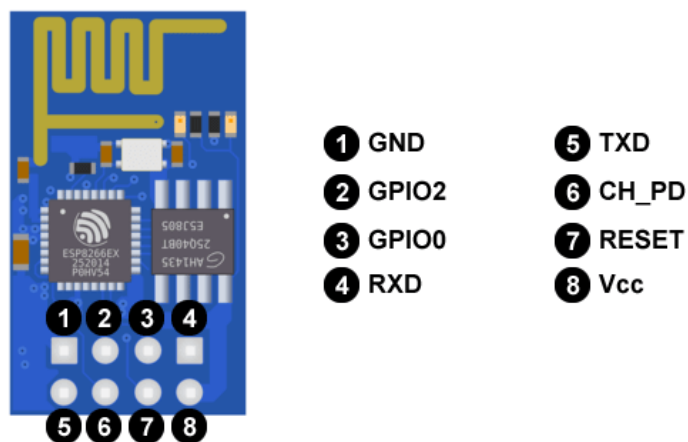


Figura 7:[Conexiones ESP01]

Otra gran desventaja del módulo ESP01 aparte de tener únicamente 4 pines para poder conectar sensores, es la ausencia de pines de entrada analógica. Por tanto, dada la limitada conectividad disponible, será necesario utilizar dispositivos con mejores prestaciones y más complejos como el ESP32.

La plataforma ESP32 ofrece a bajo costo capacidades Wifi, bluetooth y BLE. Es la evolución del microcontrolador ESP8266 mejorando sus capacidades de comunicación y procesamiento computacional. Tiene una CPU de dos núcleos de hasta 240Mhz que se pueden controlar independientemente. Además, integra internamente una gran cantidad de periféricos incluyendo: sensores táctiles capacitivos, sensor de efecto Hall, amplificadores de bajo ruido, UART e I²C.



Figura 8:[Dispositivo ESP32]

Como se observa en la Figura 9, el microcontrolador ESP-WROOM-32 [7] tiene 48 pines con múltiples funciones, aunque no todos los pines están expuestos en todas las placas de desarrollo ESP32 [8], y hay algunos pines que no se pueden usar. La placa ESP32 Node MCU [9] cuenta con 36 pines GPIO de los cuales 18 son canales de conversión analógica a digital, 3 son interfaces SPI, 3 son interfaces UART, 2 son interfaces I²C, 16 son canales de salida PWM, 2 son convertidores de digital a analógico (DAC) y 10 son GPIO de detección capacitiva.

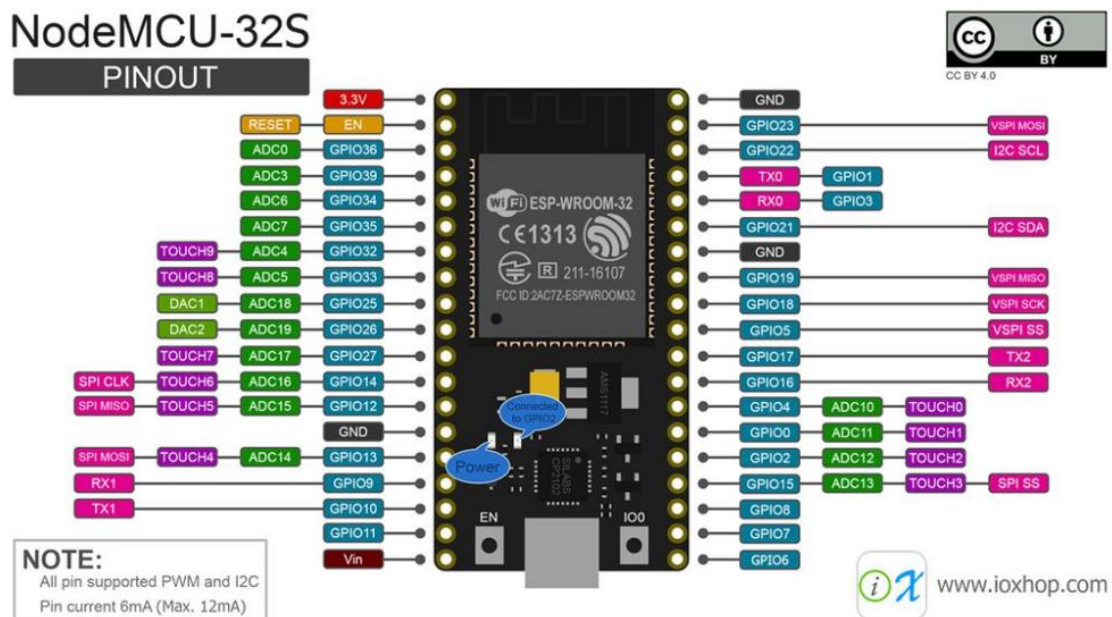


Figura 9:[Conexiones ESP32]

Las funciones ADC (convertidor analógico a digital) y DAC (convertidor digital a analógico) se asignan a pines estáticos específicos. Sin embargo, es posible también configurar la función de los mismos indicando en el código qué pines son UART, I²C, SPI, PWM, etc. Esto es posible debido a la función de multiplexación del chip ESP32. Los pines GPIO 6 a GPIO 11 están conectados al flash SPI integrado en el chip ESP-WROOM-32 y no se recomienda usar estos pines en proyectos ya que tienen funciones específicas.

4.2 Variables a monitorizar y sensores utilizados

4.2.1 Temperatura y Humedad

Para medir la temperatura y la humedad se utilizará el sensor DHT11 [10]. Este sensor es de tamaño pequeño y bajo coste, con una sensibilidad de ± 2 °C para temperatura y ± 5 % en humedad. Para la medición de la humedad y temperatura del aire emplea un sensor capacitivo de humedad y un termistor respectivamente, proporcionando los datos a través de un protocolo de comunicación parecido al One-Wire en el terminal de datos.

Este sensor es bastante simple de usar, pero requiere una sincronización cuidadosa para el bus de datos. Los principales inconvenientes de este sensor es que únicamente se pueden obtener nuevos datos como mínimo una vez cada segundo y el sensor es relativamente poco preciso. Su precio es de unos 2€ por separado, aunque también puede encontrarse montado en una PCB para facilitar su uso, ya que incluye una resistencia pull-up de 10 k Ω necesaria para el funcionamiento del bus de comunicación y un LED que nos avisa del funcionamiento

Como vemos en la *Figura 10*, el sensor DHT11 cuenta con 4 terminales (3 en el caso de PCB) que son:

- **VCC:** alimentación
- **I/O:** transmisión de datos
- **NC:** no conecta, pin al aire
- **GND:** conexión a tierra

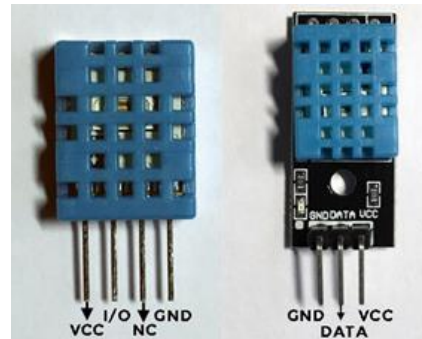


Figura 10:[Sensor DHT11]

Aunque la salida del sensor DHT11 sea digital, se trata de un dispositivo analógico y dentro del propio dispositivo se realiza todo el acondicionamiento de señal, incluyendo la conversión analógico-digital. La trama de datos que se obtiene a la salida tiene 40 bits de longitud y contiene la información de humedad y temperatura. Como podemos ver en la *Figura 11*, el primer grupo de 8 bits es la parte entera de la humedad y el segundo grupo la parte decimal. Lo mismo ocurre con el tercer y cuarto grupo, que corresponden la parte entera de la temperatura y la parte decimal. Por último, para confirmar que no haya datos corruptos hay un grupo de bits de paridad antes de finalizar la trama.

En el caso de la temperatura y la humedad, los bits 0 y 1 representan el valor en binario del dato recibido y que corresponde con un valor en decimal. Por ejemplo, en la *Figura 11*, la temperatura tiene un valor de 00011000 que corresponde a 24 °C.

<u>0011 0101</u>	<u>0000 0000</u>	<u>0001 1000</u>	<u>0000 0000</u>	<u>0100 1001</u>
8 bits humedad	8 bits humedad	8 bits temperatura	8 bits temperatura	bits de paridad

Figura 11:[Trama de datos sensor DHT11]

En *Figura 12* se muestran las características más significativas del sensor DHT11:

MODELO		DHT11	
Alimentación		de 3,5 V a 5 V	
Consumo		2,5 mA	
Señal de salida		Digital	

Temperatura		Humedad	
Rango	de 0°C a 50°C	Rango	de 20% RH a 90% RH
Precisión	a 25°C \pm 2°C	Precisión	entre 0°C y 50°C \pm 5% RH
Resolución	1°C (8-bit)	Resolución	1% RH

Figura 12:[Características sensor DHT11]

Una alternativa es el sensor DHT22 [11], ya que es muy similar al DHT11, pero con mejores prestaciones. La precisión para medir la temperatura es del 5 % de variación, como el DHT11, pero a diferencia de él, su rango de humedad va de 0 % a 100 % y cuenta con una precisión de \pm 2 %. En cuanto a la temperatura, puede medir desde los -40 °C hasta los +125 °C con más precisión (\pm 0,5 °C). Por último, la frecuencia de toma de datos también es del doble, pudiendo tomar 2 muestras por cada segundo en vez de 1 muestra por segundo del DHT11.

A pesar de que el DHT22 tiene unas prestaciones mejores, también cuenta con algunos inconvenientes en comparación con el DHT11, ya que el coste del sensor es un poco más elevado, de unos 4€, y su tamaño es el doble. En este proyecto se ha decidido tener en cuenta ambos sensores ya que, dependiendo de la situación, en algunos casos puede resultar más interesante disponer de más precisión y en otros será mejor ahorrar en coste y espacio.



Figura 13:[Sensor DHT22]

4.2.2 CO₂ y COVs

El sensor de gas CCS811 [12] es capaz de detectar un amplio rango de Compuestos Orgánicos Volátiles (COVs) y de Dióxido de Carbono Equivalente (eCO₂). Además, incluye un termistor que permite determinar la temperatura ambiente.

El sensor CCS811 está compuesto por un sensor MOX y un pequeño microcontrolador que realiza la lectura analógica del voltaje y la acondiciona para poder enviarla por I²C. La medición del Dióxido de Carbono Equivalente se realiza entre 400 y 8192 partes por millón y el de Total de Compuestos Orgánicos Volátiles entre 0 y 1187 partes por billón. Además, admite diferentes modos en cuanto a la toma de lecturas de datos: 1, 10 y 60 segundos o 250 milisegundos.

Por otro lado, el sensor incluye un regulador de 3,3V para usarse tanto a 3,3V como a 5V. Se trata de un sensor bastante compacto, con dimensiones de 21x18x3mm, y un peso de apenas 1,2 gramos.

A pesar de su pequeño tamaño, el sensor consta de 8 pines de conexión, que pueden dividirse en dos tipos según la función que desempeñan:



Figura 14:[Sensor CCS811]

- Pines de alimentación

Vin: el pin de alimentación. El sensor funciona con 3,3V, pero como ya se ha dicho antes, incluye un regulador de voltaje por lo que puede alimentarse con valores comprendidos entre 3,3 y 5 voltios de manera segura.

3V3: salida de 3,3V del regulador de voltaje y puede proporcionar hasta 100mA de corriente de salida.

GND: tierra del sensor.

- Pines lógicos

SCL: pin de reloj I²C, que va conectado a la línea de reloj I²C del dispositivo ESP32. En la placa del sensor existe ya una resistencia de pullup de 10 kΩ y puede usarse entre 3 y 5 voltios.

SDA: pin de datos I²C, que va conectado a la línea de datos I²C del dispositivo ESP32. En la placa del sensor existe ya una resistencia de pullup de 10 kΩ y puede usarse entre 3 y 5 voltios.

INT: pin de salida de interrupción. Es una lógica de 3V y puede usarse para detectar cuando una nueva lectura está lista o cuando una lectura es demasiado alta o baja.

WAKE: pin de activación para el sensor. Es necesario que se encuentre conectado a tierra para que el sensor pueda mandar datos.

RST: pin de reinicio. Cuando se conecta a tierra, el sensor se reinicia. Este pin está desplazado de nivel para que pueda usarse entre 3 y 5 voltios.

4.2.3 CO y Formaldehído

Los sensores de gas de la serie MQ son sensores analógicos electroquímicos y varían su resistencia cuando se exponen a determinados gases. Internamente posee un calentador encargado de aumentar la temperatura interna de forma que el sensor pueda reaccionar con los gases provocando un cambio en el valor de la resistencia. El calentador dependiendo del modelo puede necesitar un voltaje entre 5 y 1 voltio. El sensor se comporta como una resistencia y necesita una resistencia de carga (RL) para cerrar el circuito y con este hacer un divisor de tensión y poder leerlo desde un microcontrolador.

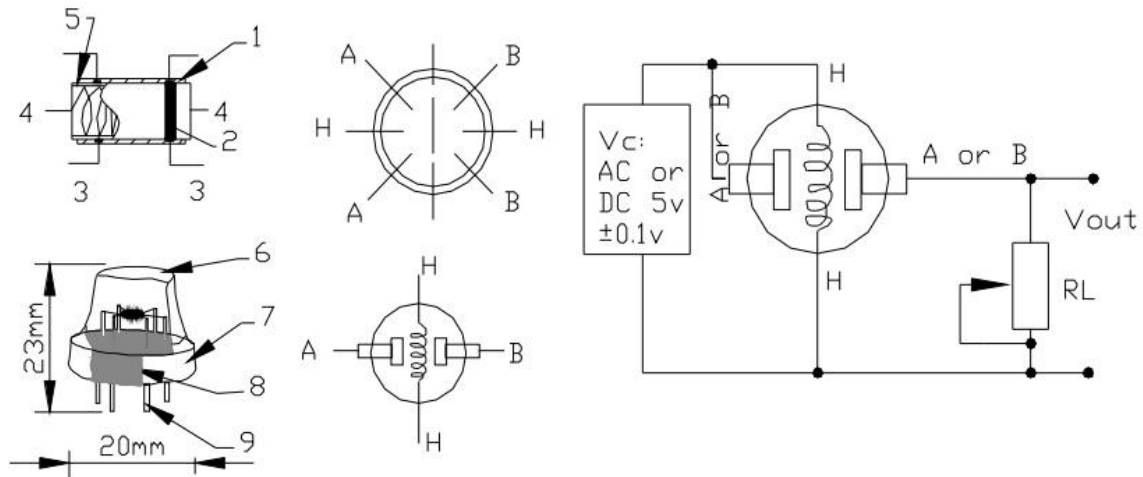


Figura 15:[Circuito interno sensor MQ]

En el mercado, generalmente los sensores MQ se encuentran en módulos, lo que simplifica el conexionado y facilita su uso, al ser únicamente necesario alimentar el módulo para poder leer la información del sensor. Estos módulos tienen una salida digital que internamente trabaja con un comparador y con la ayuda de un potenciómetro es posible calibrar el umbral para así poder interpretar la salida como presencia o ausencia del gas. También posee una salida analógica para determinar el valor exacto de la concentración de ciertos gases.



Figura 16:[Esquema de pines sensor MQ]

Dado que la relación entre la lectura analógica y el valor real no es lineal, es necesario escalar el valor leído con la ayuda del datasheet del sensor MQ y del gas correspondiente. El datasheet de cada sensor proporciona unas gráficas que permiten obtener la concentración del gas a partir de la relación entre la resistencia del sensor R_0 y la resistencia medida R_s , así como la resistencia R_L . Las gráficas emplean una en escala logarítmica para ambos ejes y, en general, son aproximadamente lineales bajo estas escalas, por lo que la concentración resultará:

$$\text{Concentración} = 10^{(A \cdot \log(R_s/R) + B)}$$

Para determinar la concentración necesitaremos la recta que la aproxima, para lo cual se utilizará la hoja de cálculo Excel. Para el proyecto, se usará el sensor MQ7 [13] capaz de detectar monóxido de carbono (CO) y el MQ138 [14] que detecta formaldehído.

○ MQ7

Este sensor es de alta sensibilidad al monóxido de carbono (CO), pero también es sensible al Hidrógeno (H_2). El sensor MQ7 necesita calentarse con una tensión alternando entre 5 y 1,4 voltios con las siguientes características:

- 5V durante 60 segundos, sin usar estas lecturas para la medición de CO.
- 1.4V durante 90 segundos, usando estas lecturas para la medición de CO.



Figura 17:[Sensor MQ7]

Para convertir el voltaje analógico de la salida del sensor a unidades correspondientes a la medición del gas, hay que escalar el valor leído. Sin embargo, como ya se ha mencionado anteriormente, la relación entre ambos parámetros no es lineal, por lo que será necesario estimar la curva a partir de la información proporcionada por el datasheet (Figura 18).

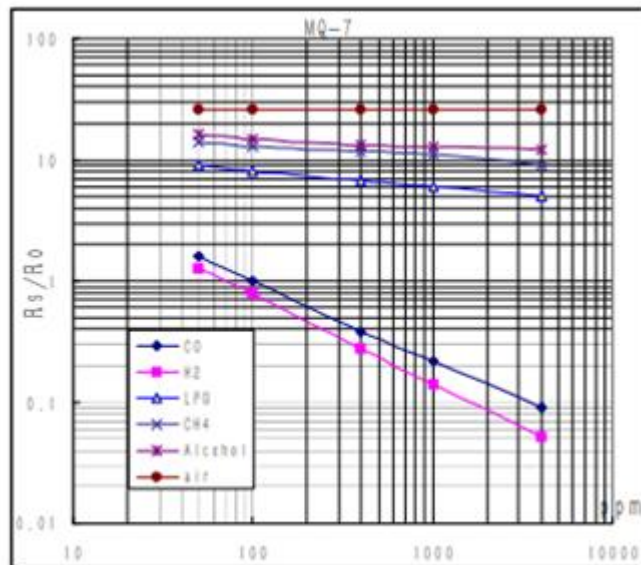


Fig 18 shows the typical sensitivity characteristics of the MQ-7 for several gases.

in their: Temp: 20°C,

Humidity: 65%,

O₂ concentration 21%

RL=10k Ω

Ro: sensor resistance at 100ppm CO in the clean air.

Rs: sensor resistance at various concentrations of gases.

Figura 18:[Curva logarítmica de sensibilidad sensor MQ7]

A partir de esta curva será necesario estimar y por regresión hallar la ecuación que relacionará ambos parámetros. Para ello se usará Excel, ingresando los datos de la curva de CO con la mayor cantidad de puntos posibles y, tras graficarlo, estimar una línea de tendencia con una ecuación potencial como podemos observar en la Figura 19.

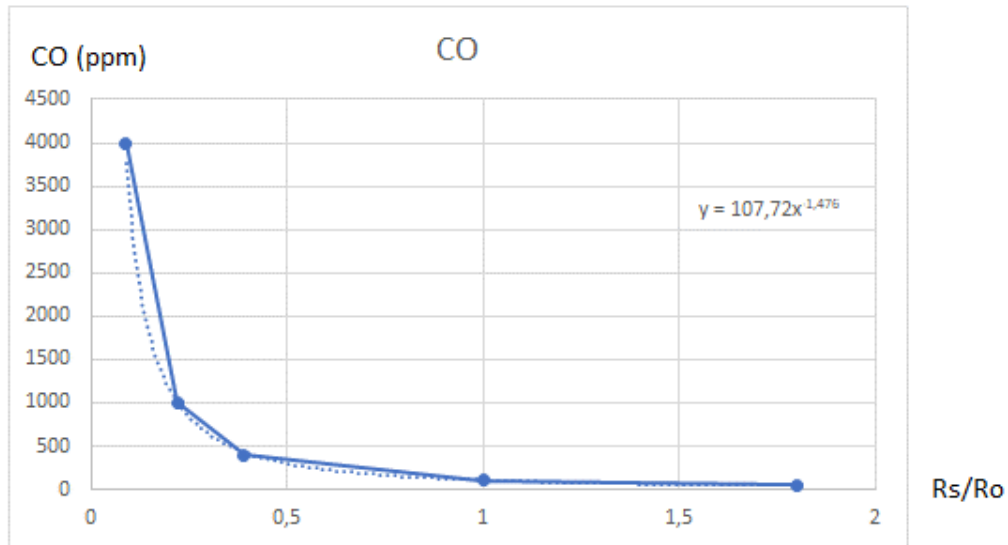


Figura 19:[Curva de sensibilidad sensor MQ7]

La ecuación obtenida a partir de los datos es:

$$CO \text{ (ppm)} = 107.72 * (R_s/R_o)^{1.476}$$

donde R_o es una constante que equivale al valor de la resistencia del sensor cuando se lo expone a una concentración de 100 ppm y R_s es la resistencia del sensor. Para calcular el valor de R_s se despeja de la ecuación el divisor de voltaje que forma el sensor con la resistencia de carga R_L , que en nuestro caso es de 1 kΩ:

$$Voltaje = 1.4 * \frac{R_L}{R_L + R_s}$$

Y con esto:

$$R_s = R_L * \frac{1.4 - Voltaje}{Voltaje}$$

Por tanto, sabiendo la ecuación para calcular R_s y que el valor de R_o es un valor constante que hay que calibrar, la ecuación para obtener la concentración de CO a partir del voltaje de salida es la siguiente:

$$CO \text{ (ppm)} = 107.72 * ((R_L * (1.4 - Voltaje))/(Voltaje * R_o))^{1.476}$$

○ MQ138

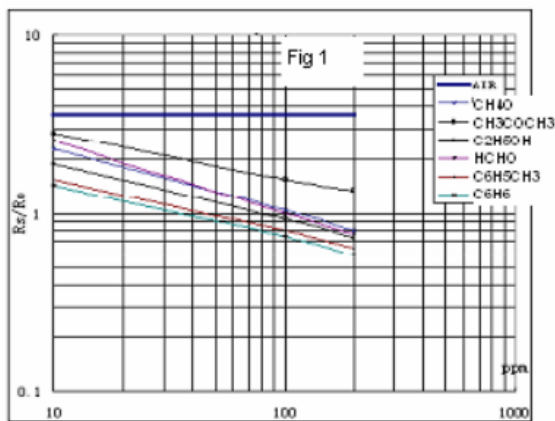
Estos sensores se utilizan en equipos de control de calidad del aire para edificios y oficinas y son adecuados para la detección de formaldehído (CH_2O) entre otros gases. Este sensor necesita calentarse con una tensión continua de 5 V para poder obtener medidas precisas.

Al igual que con el sensor anterior, para disponer de los valores en unidades correspondientes a la medición de formaldehído será necesario el valor leído. Sin embargo, para este sensor la relación entre la lectura analógica y el valor real tampoco es lineal, por lo que, al igual que antes, será necesario estimar la curva proporcionado por el datasheet (Figura 21).



Figura 20:[Sensor MQ138]

Sensitivity Characteristics



Influence of Temperature/Humidity

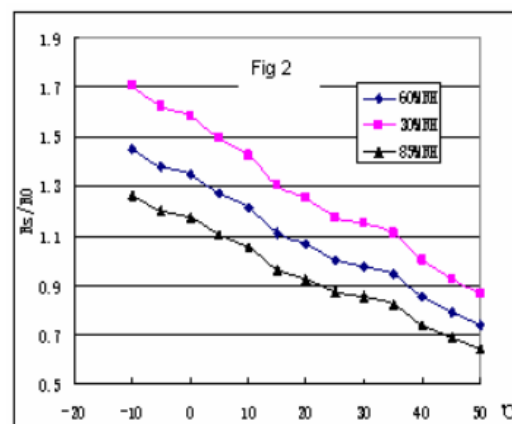


Figura 21:[Curva logarítmica sensibilidad e influencia de la temperatura sensor MQ138]

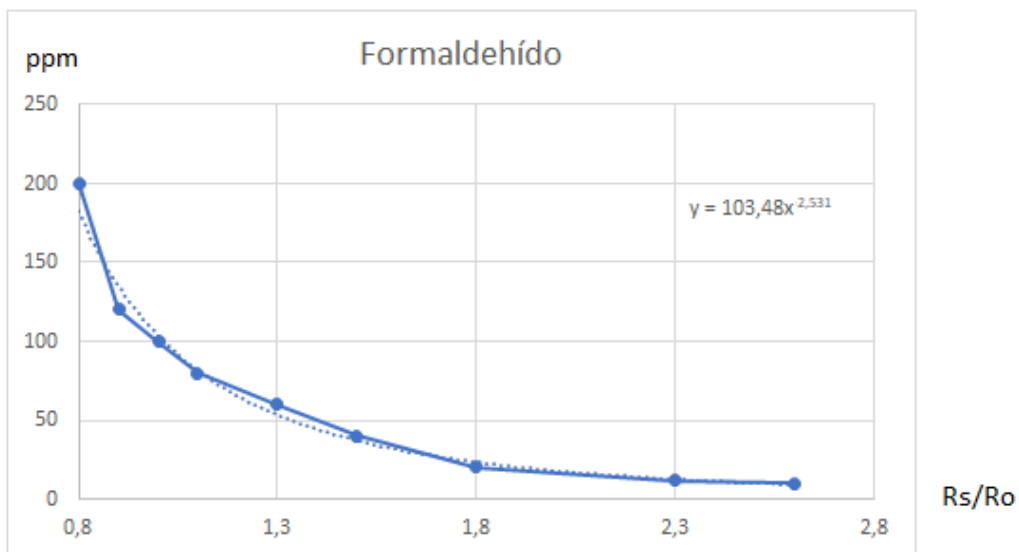


Figura 22:[Curva sensibilidad sensor MQ138]

La ecuación obtenida a partir de los datos de la *Figura 22* es:

$$\text{Formaldehído (ppm)} = 103.48 * (Rs/Ro)^{2.531}$$

donde Ro es una constante que equivale al valor de la resistencia del sensor cuando se lo expone a una concentración de 100 ppm de Tolueno y Rs es la resistencia del sensor.

Para calcular el valor de Rs hay que despejar la ecuación del divisor de voltaje que forma el sensor con la resistencia de carga RL que en nuestro caso es de 1 kΩ:

$$\text{Voltaje} = 5 * \frac{RL}{RL + Rs}$$

Entonces:

$$Rs = RL * \frac{5 - \text{Voltaje}}{\text{Voltaje}}$$

Por tanto, sabiendo la ecuación para calcular Rs y que el valor de Ro es un valor constante que hay que calibrar, la ecuación para obtener la concentración de CO a partir del voltaje de salida es la siguiente:

$$\text{Formaldehído (ppm)} = 103.48 * ((RL * (5 - \text{Voltaje})) / (\text{Voltaje} * Ro))^{2.531}$$

4.2.4 Temperatura del agua

El DS18B20 [15] es un sensor digital de temperatura fabricado por Maxim Integrated que utiliza el protocolo One-Wire para comunicarse. Este protocolo necesita solo un pin de datos para comunicarse y permite conectar más de un sensor en el mismo bus. Puede encontrarse en dos formatos: uno de tamaño pequeño, con el encapsulado tipo TO-92 similar al empleado en transistores pequeños y otro, el más comercial, dentro de un tubo de acero inoxidable resistente al agua.

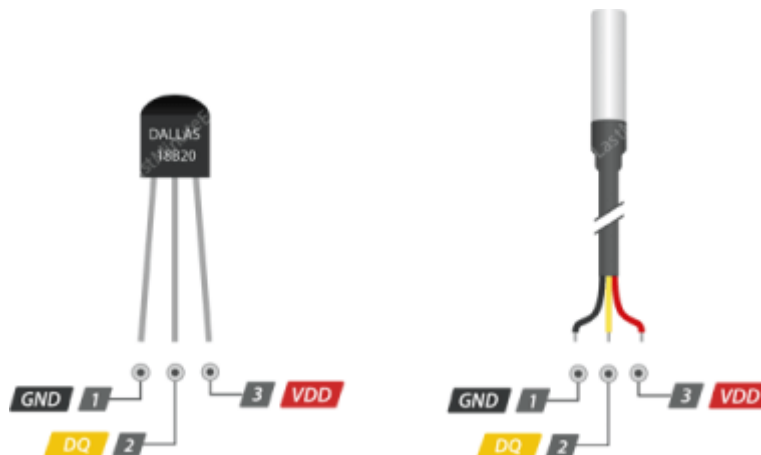


Figura 23:[Sensor DS18B20]

Con este sensor se puede medir temperatura desde los $-55\text{ }^{\circ}\text{C}$ hasta los $125\text{ }^{\circ}\text{C}$ con una resolución programable desde 9 bits hasta 12 bits. Para temperaturas entre $-10\text{ }^{\circ}\text{C}$ y $85\text{ }^{\circ}\text{C}$ el sensor tiene una desviación de $\pm 0,5\text{ }^{\circ}\text{C}$ y para el resto de las temperaturas (entre $-55\text{ }^{\circ}\text{C}$ y $125\text{ }^{\circ}\text{C}$) el error es de $\pm 2\text{ }^{\circ}\text{C}$. Cada sensor tiene una dirección única de 64 bits establecida de fábrica que sirve para identificar al dispositivo con el que se está comunicando.

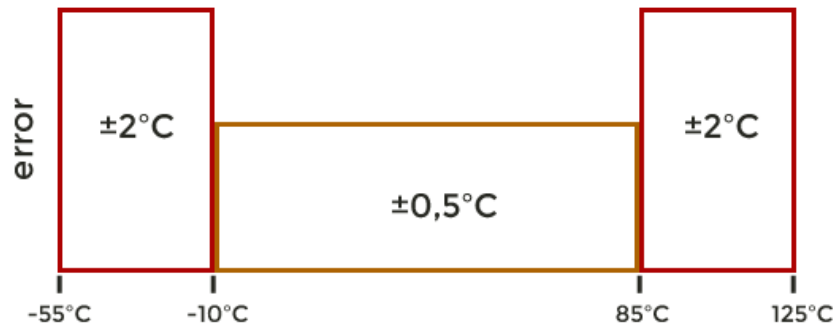


Figura 24:[Error de desviación sensor DS18B20]

Gracias a que el sensor DS18B20 se comunica mediante el protocolo One-Wire, se consiguen dos cosas: por un lado, robustez en la transmisión de los datos ya que trabaja con datos digitales, mucho menos sensibles a los efectos adversos del ruido que las señales analógicas. Y, por otro lado, permite conectar muchos sensores de temperatura con un único pin digital, al ser un protocolo con direccionamiento.

El sensor DS18B20 cuenta con tres pines:

- **VDD:** entrada de tensión de alimentación. Es posible alimentarlo desde 3V a 5,5V.
- **GND:** toma de tierra. A este pin se le conectará la referencia 0V del circuito.
- **DQ:** pin de datos. Por este pin es por donde se recibirán todos los datos en el protocolo One-Wire.

Una de las características más interesantes de este sensor es que puede trabajar con diferentes resoluciones. El DS18B20 admite resoluciones de 9 bits ($0,5\text{ }^{\circ}\text{C}$), 10 bits ($0,25\text{ }^{\circ}\text{C}$), 11 bits ($0,125\text{ }^{\circ}\text{C}$), y 12 bits ($0,0625\text{ }^{\circ}\text{C}$), aunque por defecto utiliza la resolución de 12 bits.

RESOLUCIÓN	TEMPERATURA
9-bit	$0,5^{\circ}\text{C}$
10-bit	$0,25^{\circ}\text{C}$
11-bit	$0,125^{\circ}\text{C}$
12-bit	$0,0625^{\circ}\text{C}$

Figura 25:[Tabla de resoluciones del DS18B20]

Con todas estas características, el DS18B20 se convierte en un sensor barato (entre 1€ y 3€) y bastante potente, con unas capacidades superiores a otros en el mismo rango de precios. En la *Figura 26* se muestran las características más relevantes de este tipo de sensores:

CARACTERÍSTICA	VALOR
Voltaje de alimentación	3V a 5,5V
VDD	voltaje de alimentación
GND	Tierra
DQ	Datos
Rango de temperaturas	-55°C a 125°C
Error (-10°C a 85°C)	±0,5°C
Error (-55°C a 125°C)	±2°C
Resolución programable	9-bit, 10-bit, 11-bit o 12-bit (default)

Figura 26:[Características sensor DS18B20]

4.2.5 Conductividad

El sensor analógico DFR0300 [16] se utiliza para medir la conductividad eléctrica en una solución acuosa, siendo muy empleado para evaluar la calidad del agua en acuicultura, medio ambiente y otras áreas. En el Sistema Internacional de Unidades, la unidad de conductividad es Siemens partido por metro (S/m).

El sensor consta de dos partes: una sonda de conductividad con conector BNC y una tarjeta electrónica que se encarga de acondicionar la medida que obtiene la sonda para que un microcontrolador la pueda monitorizar.

La conductividad es la capacidad que tiene un material de dejar pasar a través de él la corriente eléctrica. El agua pura, H₂O, no conduce la electricidad. Sin embargo, prácticamente toda el agua con la que estamos en contacto (en el grifo, mineral, lluvia, mar...) no es agua pura, sino que es agua con una disolución de sales en diferente concentración.

El valor que se da de la conductividad de una disolución está referenciado a 25 °C. Las unidades de medida para la conductividad son en unidad de resistencia/unidad de longitud. La conductividad eléctrica de un líquido se mide determinando la resistencia de la solución entre dos electrodos planos o cilíndricos separados por una distancia fija.



Figura 27:[Sensor DFR0300]

La definición de resistencia es:

$$R = \rho \frac{L}{A}$$

Donde ρ es la resistividad, L es la longitud entre los electrodos de la sonda y A es el área transversal de estos. Por lo que, por definición, podemos definir la conductividad como:

$$\kappa = \frac{1}{\rho} \rightarrow \kappa = \frac{1}{R} * \frac{L}{A}$$

Para obtener el valor de la conductividad a la salida del sensor en voltios, este dispone de un circuito con el objetivo de calcular una tensión de salida a partir de la resistencia del electrodo. La función es la siguiente:

$$V_{out} = \frac{R_{10}}{R} * |V_{in}|$$

donde V_{in} tiene un valor aproximado de 200mV y R_{10} de 820 Ω . Por tanto, la ecuación de la conductividad del agua en función del voltaje de salida del sensor será la siguiente:

$$\kappa = \frac{Q}{R_{10} * |V_{in}|} * V_{out}$$

donde Q es el caudal de agua que atraviesa la sonda y equivale a L/A . De esta forma es posible calcular el valor de la conductividad asumiendo una temperatura constante de 25 °C. Para tener en cuenta las variaciones de la misma en el cálculo será necesario hacer un ajuste del voltaje en función de la temperatura real del agua:

$$Coef.Temperatura = 1 + 0.0185 * (Temperatura_{Real} - 25)$$

$$V_{out} = V_{Salida} / Coef.Temperatura$$

El sistema de acondicionamiento que transforma la medida de conductividad a milivoltios consta únicamente de tres pines:

- **VDD:** pin de alimentación, operando a un voltaje de 5 voltios.
- **GND:** pin de tierra.
- **AQ:** pin analógico en el que se encuentra el voltaje de salida tras el acondicionamiento.



Figura 28:[Sistema acondicionamiento sensor DFR0300]

En la tabla *Figura 29* pueden observarse las principales características de este sensor.

VOLTAJE DE FUNCIONAMIENTO:	5V DC.
TAMAÑO DE PCB:	45x32 mm
INTERVALO DE MEDIDA	1 ms / cm - 20 ms / cm
RANGO DE MEDICIÓN	0-14
TEMPERATURA DE FUNCIONAMIENTO	5-40 °C
PRECISIÓN	<± 10% F.S (usando el ADC de 10 bits Arduino)
ELECTRODO	electrodo constante K = 1, conector BNC
INDICADOR DE ENCENDIDO	LED
LONGITUD DEL CABLE DEL ELECTRODO	unos 60 cm
CONECTOR SENSOR	Conector BNC
CONECTOR INTERFAZ	Conector de 3 pines
DIMENSIONES DEL ADAPTADOR:	43mm × 32mm

Figura 29:[Características sensor DFR0300]

4.2.6 Control de pH

El sensor de Potencial de Hidrógeno (pH) es un transductor que permite conocer el pH de una solución. Esto se consigue gracias a que el sensor mide la diferencia de potencial eléctrico entre un electrodo de pH y un electrodo de referencia. Esta diferencia de potencial eléctrico se relaciona con la acidez siendo la solución más ácida cuanto menos pH tenga.

El sensor seleccionado en este proyecto es el pH SEN0161 [17] ya que permite una rápida solución para diseños de bajo costo sin que se sacrifique la operatividad del diseño. El sensor consta de dos elementos:

- Sonda para medir el pH del agua a través de la detección de una pequeña corriente eléctrica generada por la actividad de los iones de hidrógeno.
- Placa de acondicionamiento que permite su conexión directa a un sistema electrónico, en nuestro caso, la placa ESP32.



Figura 30:[Sensor de pH SEN01061]

La salida del sensor de pH está en el rango de los milivoltios y puede establecerse una relación entre el pH medido en el agua y la salida, siempre que la temperatura ambiente sea constante y de 25 °C. En *Figura 31* podemos observar dicha relación:

VOLTAGE (mV)	pH value	VOLTAGE (mV)	pH value
414.12	0.00	-414.12	14.00
354.96	1.00	-354.96	13.00
295.80	2.00	-295.80	12.00
236.64	3.00	-236.64	11.00
177.48	4.00	-177.48	10.00
118.32	5.00	-118.32	9.00
59.16	6.00	-59.16	8.00
0.00	7.00	0.00	7.00

Figura 31:[Relación voltaje y valor de pH]

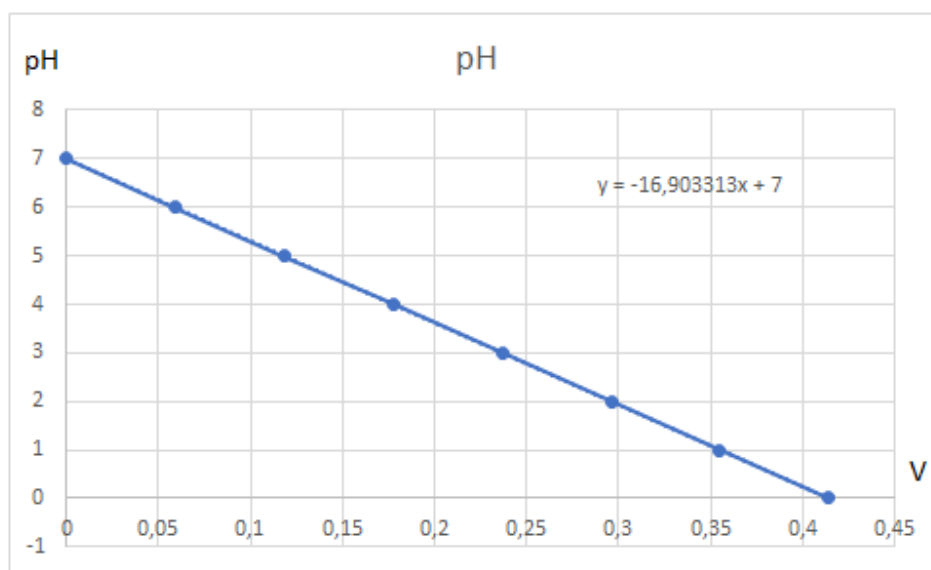


Figura 32:[Relación de conversión de voltaje a pH]

Interpolando linealmente los datos de la hoja de características de la Figura 31, la ecuación que se obtiene es:

$$pH = -16.903313 * Voltaje + 7$$

El circuito de acondicionamiento que transforma la medida de pH a voltios, conocido como conector interfaz pH 2.0, tiene un tamaño pequeño y consta de tres pines:

- **VDD:** es la tensión de alimentación, es decir, que voltaje necesita para que el sensor de pH funcione correctamente.
- **GND:** pin con la toma de tierra.
- **AQ:** salida analógica después del acondicionamiento por el que se transmite la información.

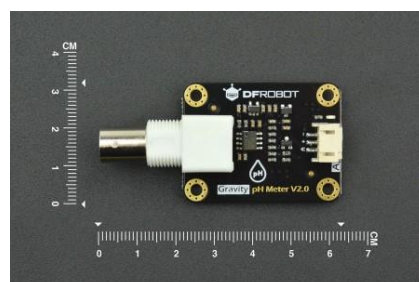


Figura 33:[Sistema acondicionamiento sensor SEN0161]

En la *Figura 34* pueden observarse las principales características de este sensor.

VOLTAJE DE FUNCIONAMIENTO:	5V DC.
CORRIENTE DE FUNCIONAMIENTO:	
TIEMPO DE RESPUESTA:	≤ 1min
RANGO DE MEDICIÓN	0-14
PRECISIÓN	± 0.1pH (25 °C)
POTENCIOMETRO	Potenciómetro de ajuste de ganancia
INDICADORES	Indicador de encendido LED
MEDICIÓN DE LA TEMPERATURA	0-60 °C
CONECTOR SENSOR	Conector BNC
CONECTOR INTERFAZ PH 2.0	Conector de 3 pines
DIMENSIONES DEL ADAPTADOR:	43mm × 32mm

Figura 34:[Características sensor SEN0161]

4.3 Conexiones

4.3.1 Pin GPIO

GPIO (General Purpose Input/Output) es un pin genérico usado en microcontroladores, cuyo comportamiento (incluyendo si es un pin de entrada o salida) se puede controlar (programar) por el usuario en tiempo de ejecución. Los pines GPIO no tienen ningún propósito especial definido, y no se utilizan de una forma predeterminada. La idea es que a veces, para el diseño de un sistema completo, es útil contar con un conjunto de líneas de control adicionales, y tenerlas a disposición ahorra el tiempo de tener que organizar circuitos adicionales para proporcionarlos.

Como ya hemos visto antes, la placa ESP32 Node-MCU consta de 36 pines de propósito general (GPIO) de los cuales solo podemos utilizar 30, ya que el resto (del GPIO6 al GPIO11) están reservados para funciones específicas y es mejor no utilizarlos. A continuación, se va a estudiar la conexión de los diferentes sensores a la misma.

○ Sensor DHT11 y DHT22

El proyecto emplea dos sensores digitales para medir la temperatura y la humedad: el DHT11 y el DHT22. Aunque estos dos sensores no se van a utilizar juntos en un mismo dispositivo, se les va a asignar distintos pines para asegurar el funcionamiento de ambos en el caso de que se quiera usar uno u otro. El sensor DHT11 va a ir conectado el pin GPIO12 y el DHT22 al pin GPIO12. Como ya se ha comentado antes, ambos precisan de una resistencia de 10 kΩ conectada entre el pin GPIO y la alimentación (Vcc), aunque los sensores montados en una PCB ya la tienen incluida.

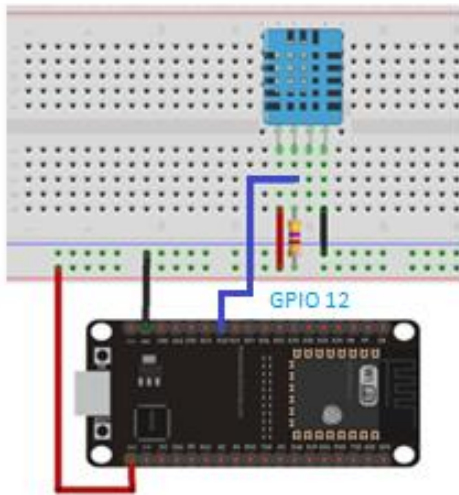


Figura 35:[Montaje sensor DHT11]

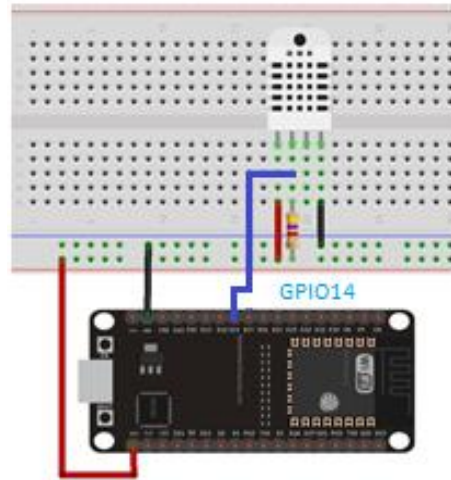


Figura 36:[Montaje sensor DHT22]

- Sensor DS18B20

Este sensor se utiliza para medir la temperatura del agua protegiendo los cables con un recubrimiento. Se puede conectar a cualquier pin GPIO, pero en este proyecto está conectado al GPIO4. Además, necesita una resistencia de 4,7 k Ω conectada entre el pin GPIO y la alimentación (Vcc).

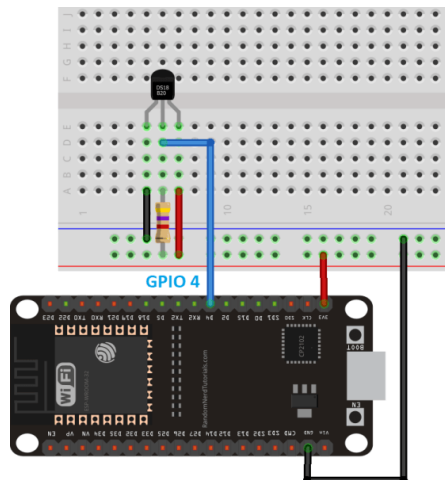


Figura 37:[Montaje sensor DS18B20]

4.3.2 Bus I²C

Un bus I²C tiene la propiedad de que un dispositivo puede controlar un conjunto de dispositivos periféricos con sólo dos pines (Input/Output) y un software muy simple. Aunque es más lento que los sistemas de bus más nuevos, I²C es el método más empleado en la actualidad para los sistemas que no necesitan ser rápidos debido a su bajo coste. I²C es un bus con uno o varios maestros, lo que significa que se pueden conectar varios microcontroladores al mismo bus y que todos ellos pueden actuar como maestro sólo con iniciar la transferencia de datos.

Este bus es usado para la transmisión de datos de control y configuración de los esclavos, como, por ejemplo, para control de volumen, conversor de señal analógica-digital o digital-analógica con baja tasa de frecuencia de muestreo, relojes a tiempo real o pequeños espacios de memoria. En este proyecto, utilizaremos el bus I²C para tomar datos de sensores electrónicos que integran un convertidor analógico-digital.

El ESP32 tiene dos canales I²C y cualquier pin se puede configurar como SDA o SCL. El pin SDA funciona como transmisor de datos y el SCL funciona como reloj. En este proyecto se utilizarán los pines que vienen por defecto: el GPIO 21 como SDA y el GPIO 22 como SCL.

○ Sensor CCS811

El sensor CCS811 tiene 8 pines de los cuales solo hace falta conectar 5 de ellos. Dos de estos pines son los pines de alimentación, que irán a Vcc y a GND, y otros dos pines se encargarán de la transmisión de datos, serán SDA y SCL. Hay que conectar un pin más, el pin WAKE, que irá conectado a GND y se encargará de mantener el sensor en estado activo.

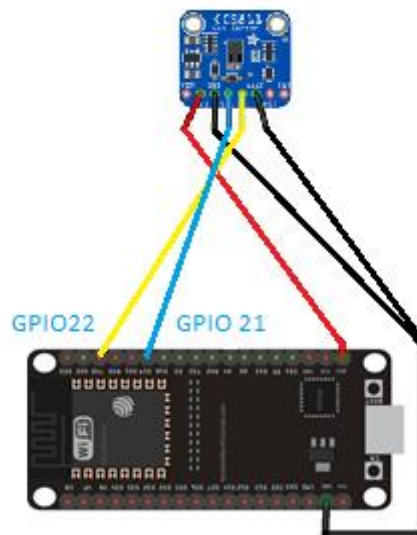


Figura 38:[Montaje sensor CSS811]

4.3.3 Señales Analógicas

Una señal analógica es una magnitud que puede tomar cualquier valor dentro de un intervalo acotado entre tierra y Vcc. Por norma general en los autómatas las entradas analógicas son más escasas, más lentas y caras que las entradas digitales.

Para entender la precisión de una entrada analógica es necesario entender cómo funciona el conversor analógico digital (ADC), que es su componente fundamental. La conversión analógico-digital es un proceso de cuantización en la cual una señal analógica es representada por su equivalente en estados binarios. La resolución de un conversor indica el número de valores discretos que este puede producir sobre un rango de valores de voltaje y generalmente es expresado en bits. Por ejemplo, un conversor que codifica una entrada analógica de 256 valores discretos [0...255] tiene una resolución de 8 bits, puesto que 256 equivale a 2⁸.

El ESP32 tiene 16 canales de entrada ADC con una resolución de 12 bits. Esto significa que puede obtener lecturas analógicas que van de 0 a 4095, de las cuales 0 corresponde a 0V y 4095 a Vcc. Como se puede observar en la *Figura 40*, los pines ESP32 ADC no tienen un comportamiento lineal cuando los valores son muy bajos o muy altos, es decir, probablemente no se podrá distinguir entre 0 y 0.1V, o entre 3.2 y 3.3V.

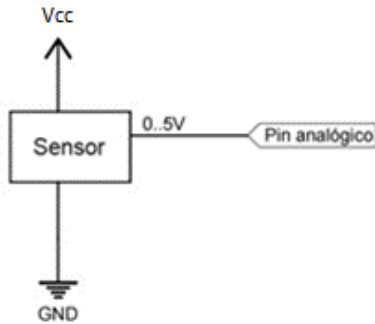


Figura 39:[Esquema sensor analógico]

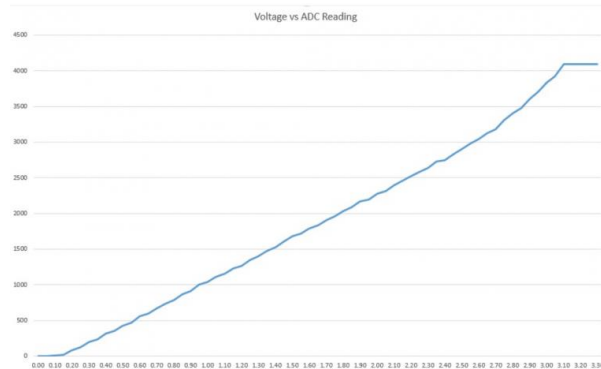


Figura 40:[Comportamiento sensor analógico]

De las entradas analógicas disponibles en el ESP32 se usarán únicamente cuatro para los sensores de CO, formaldehído, conductividad del agua y pH. Evidentemente no se van a utilizar los cuatro sensores a la vez en un mismo dispositivo puesto que unos están pensados para procesos donde se utilicen compuestos químicos, como la inyección de plásticos (CO, formaldehído), y los otros están pensados para medir las condiciones del agua (conductividad, PH). Sin embargo, se ha optado por asignar un pin de entrada diferente a cada uno para simplificar su configuración.

○ Sensor MQ7

El sensor MQ7 tiene 4 pines de conexión, pero en este proyecto solo se usarán tres de ellos: el de alimentación, el de tierra y el analógico. Para calentar el sensor, se utilizará pin GPIO5, que se alimentará durante 60 segundos a 5 voltios y durante 90 segundos a 1,4 voltios. La entrada analógica irá conectada al pin GPIO34 (ADC6) y hay que acondicionarla a valores de concentración.

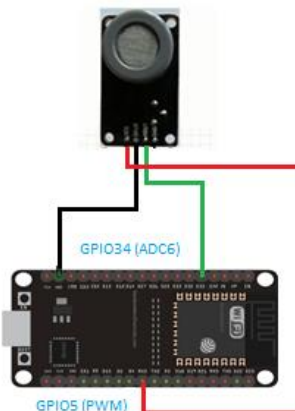


Figura 41:[Montaje sensor MQ7]

○ Sensor MQ138

El sensor MQ138 tiene tres pines que van conectados a la alimentación de 5 voltios, a tierra y a la entrada GPIO35 (ADC7) que habrá que acondicionar a valores de concentración.

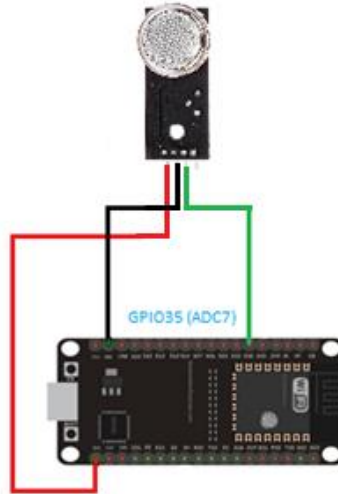


Figura 42:[Montaje sensor MQ138]

○ Sensor DFR0300

El sensor DFR0300 tiene una tarjeta de acondicionamiento con tres pines de conexión. Dos de ellos son los dedicados a la alimentación y van conectados a 5V y a tierra. El pin restante es un pin analógico conectado a GPIO32 (ADC4) y es necesario acondicionar la señal a valores de conductividad.

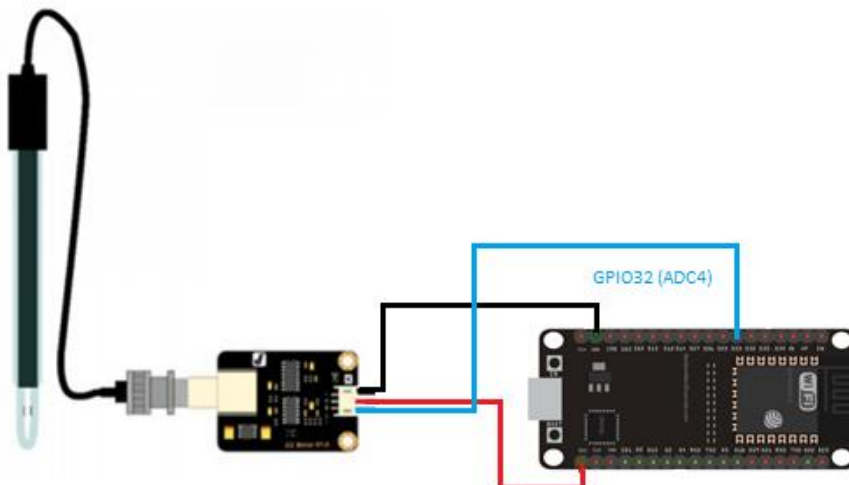


Figura 43:[Montaje sensor DFR0300]

○ Sensor de pH SEN 0161

El sensor de pH SEN 0161 cuenta con una sonda con tres pines con conexión a alimentación, a tierra y a un pin analógico en nuestro caso el GPIO33 (ADC5). Esta señal analógica hay que transformarla a valores de pH.

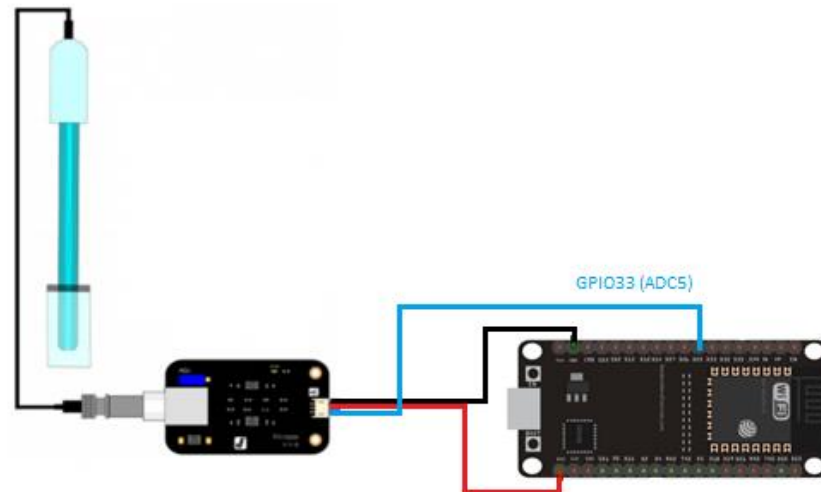


Figura 44:[Montaje sensor SEN0161]

4.4 Esquema

Tras haber hecho un análisis de cómo funcionan los dispositivos ESP32 y haber explicado al detalle las distintas posibilidades en cuanto a sensores para poder medir parámetros externos, en la *Figura 45* se puede ver un esquema de las conexiones de todos los sensores.

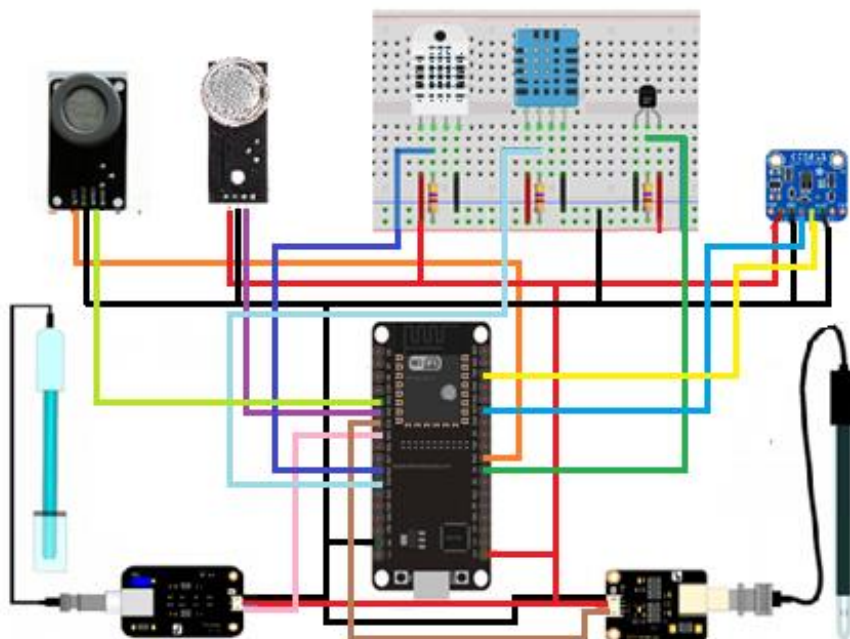


Figura 45:[Montaje completo proyecto]

Evidentemente, en la práctica no vamos a ver un dispositivo con todos estos sensores conectados, ya que unos sensores están pensados para utilizarse en procesos donde se escapan gases contaminantes y otros para la medición de parámetros de aguas residuales. En *Figura 46*, podemos observar el montaje de un dispositivo ESP32 con los sensores analizados anteriormente, donde cabe destacar la ausencia de los sensores de pH y de conductividad debido a que el pedido se ha retrasado por el coronavirus.

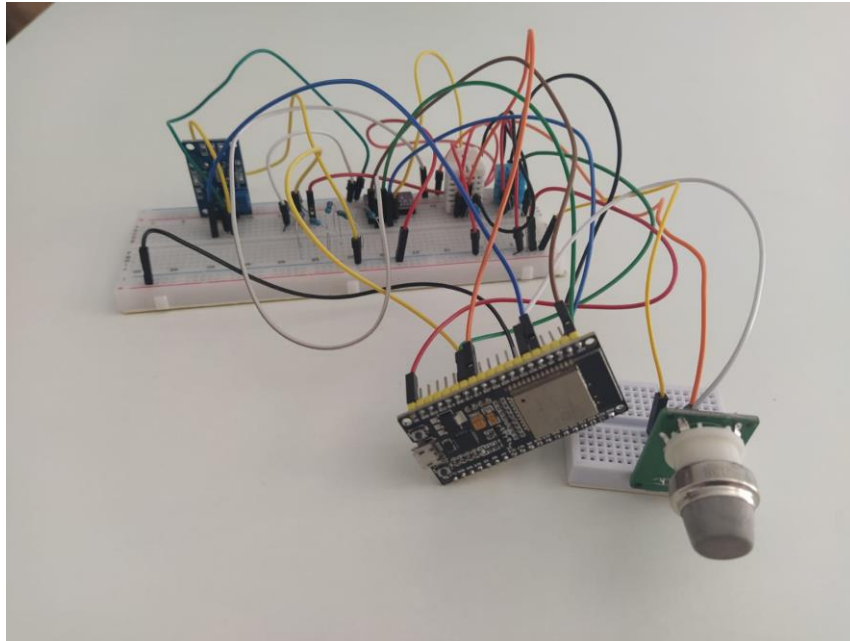


Figura 46:[Montaje real del proyecto]

5. Software

5.1 Instalación del proyecto

5.1.1 Máquina Virtual

Una máquina virtual es un software que crea una capa independiente donde se emula el funcionamiento de un ordenador real con todos los componentes de hardware que necesita para funcionar (disco duro, memoria RAM, tarjetas de red, tarjeta gráfica, etc.). Permite ejecutar cualquier sistema operativo o programa, tal y como lo haría un ordenador real, sin que nada de lo que suceda en el interior afecte al ordenador huésped que la ejecuta.

Existen dos tipos de máquinas virtuales diferenciadas por su funcionalidad:

- **Máquinas virtuales de sistema:** son las más conocidas por la mayoría de los usuarios ya que son ideales para virtualizar sistemas operativos completos.
- **Máquinas virtuales de procesos:** son más habituales en servidores y únicamente se virtualizan determinados procesos o servicios y no el sistema operativo completo.

En el proyecto, se emplearán máquinas virtuales de sistema ya que la idea es montar todo el sistema en un servidor. Actualmente existen muchos hipervisores para crear máquinas virtuales, habiéndose optado en este proyecto por VMWare Workstation [18] para su implementación. Como sistema a virtualizar se usará la distribución Linux Ubuntu Server, ya que es un sistema totalmente gratuito y optimizado para su uso en servidores.

Por defecto la máquina virtual está configurada en modo NAT, modo en el cual funciona redireccionando los puertos del router. Esto va a dificultar la implementación ya que es importante que la máquina se conecte de forma independiente a la red y no a través del host donde está instalada. Para conseguir esto hay que configurar la máquina virtual en modo BRIDGE y así se conseguirán conexiones independientes, es decir, que la máquina virtual tenga una dirección IP que se encuentra en el rango de direcciones de la red.

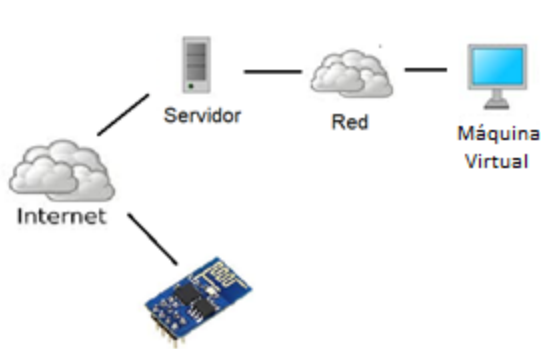


Figura 47:[Configuración NAT]

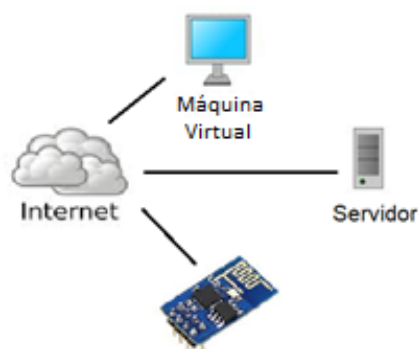


Figura 48:[Configuración BRIDGE]

5.1.2 Dockers

Docker [19] es un proyecto de código abierto para automatizar la implementación de aplicaciones en contenedores portátiles y autosuficientes que se pueden ejecutar en la nube o localmente. Para instalar los programas que integrarán el proyecto, se utilizarán imágenes Docker [20] que es una manera de empaquetar una aplicación o un servicio e implementarlo de forma confiable y reproducible.

La principal ventaja de los contenedores es que consumen muchos menos recursos (no necesitan un sistema operativo completo), se inician rápidamente y son fáciles de implementar. Esto permite tener una mayor densidad, lo que significa que se pueden ejecutar más servicios en la misma unidad de hardware, reduciendo así los costos. Además, el motor de contenedor mantiene los contenedores aislados, pero les permite compartir recursos a otros contenedores o incluso con el exterior.

En este proyecto, Docker va a estar instalado en máquinas virtuales que se van a conectar de manera independiente a la red por lo que tendrán una dirección IP específica. Al instalar Docker en nuestra máquina virtual, podemos conectar los contenedores de tres maneras distintas:

- **Bridge:** Es la que viene por defecto. Esta configuración la tendrán todos los contenedores salvo que indiquemos lo contrario en su creación y se encarga de crear un puente entre los contenedores y el Docker0.
- **Host:** utilizada por contenedores que no se aíslan de la máquina host donde se encuentra instalado Docker. Esto significa que toma todas las interfaces de red configuradas en la máquina host.
- **None:** Serán contenedores aislados, sin interfaz de red.

A parte de tener los contenedores con los programas instalados en ellos, se dispondrá de un contenedor, el Docker0, que se genera en la instalación con dirección IP por defecto 172.17.0.1 y que hará de puente entre los contenedores y el exterior o entre un contenedor y otro, al usar la configuración bridge.

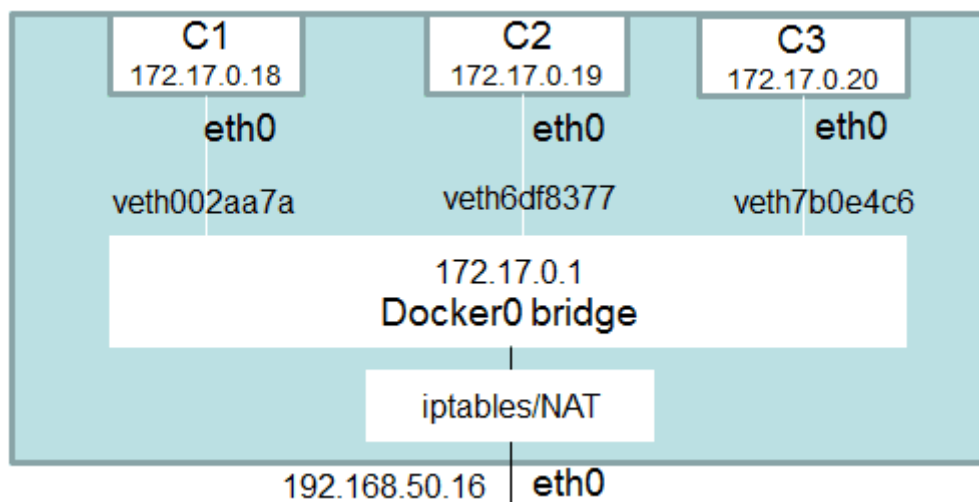


Figura 49:[Esquema estructura Docker]

5.2 Programación de los dispositivos

5.2.1 Servidor DHCP

DHCP (Dynamic Host Configuration Protocol) es un protocolo de red utilizado en redes IP para proporcionar automáticamente los parámetros de una interfaz de red, tales como dirección IP o puerta de acceso, a cada host en una red. La razón principal por la que se necesita un servidor DHCP [21] en este proyecto es para simplificar la administración de las direcciones de los diferentes nodos remotos, ya que no puede haber dos dispositivos con la misma dirección IP, y configurarlos manualmente puede generar errores. Por tanto, se ha optado por automatizar este proceso de asignación de direcciones IP para hacerlo más fácil para los usuarios y el administrador de la red.

Para facilitar las cosas a la hora de administrar el servidor DHCP, se ha instalado un contenedor con interfaz web. Esta página está alojada en el puerto 3001 y nos permite tener acceso a información de las redes y subredes, así como configurar alarmas en el caso de que una red se sature, es decir, que no pueda asignar más direcciones IP porque ya ha alcanzado el máximo permitido. Además, mediante autenticación de usuario se puede cambiar la configuración del servidor desde la interfaz web.

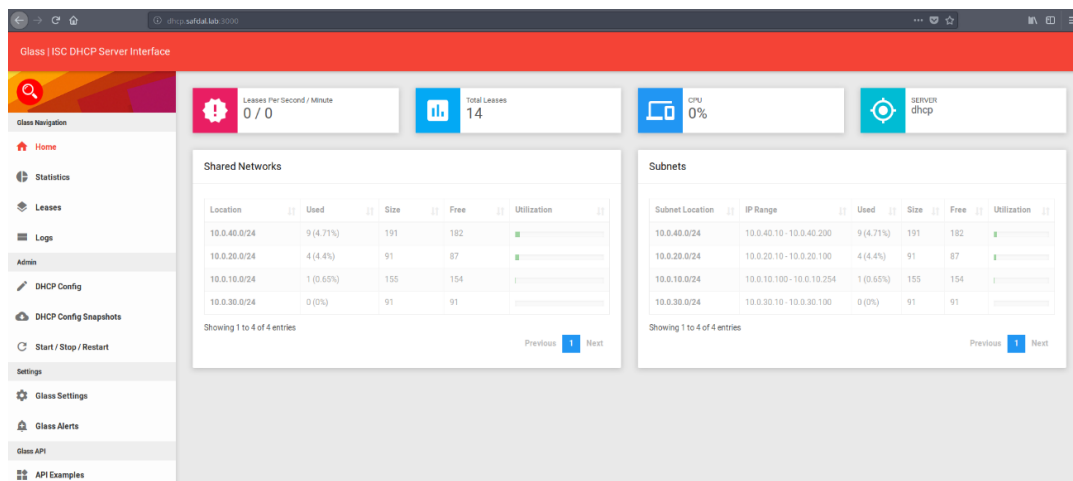


Figura 50:[Interfaz web servidor DHCP]

Para configurar el servidor, primero hay que definir el rango de direcciones que va a tener disponible, por ejemplo, de la 192.168.1.100 a la 192.168.1.199. Esto nos dará 100 direcciones disponibles para asignar a los distintos dispositivos que se conecten a la red.

Posteriormente será necesario decidir cómo va a asignar las IPs el servidor, existiendo dos opciones posibles:

- **Estática:** a cada dispositivo se le asigna siempre la misma dirección. Es la que se utilizará ya que interesa que cada dispositivo tenga una dirección única para facilitar las actualizaciones del firmware vía OTA.
- **Dinámica:** asigna dinámicamente una IP al cliente que lo solicita durante un tiempo determinado.

5.2.2 ESPHome

Debido al elevado número de dispositivos que se desea conectar a la red, la programación de los dispositivos uno a uno no es factible, requiriendo el uso de sistemas automatizados que permitan el despliegue de firmware a través de OTA para actualizar todos los nodos. Existen muchos programas para programar el firmware de dispositivos, pero los más usados son las siguientes cuatro aplicaciones:

- ESPEasy

ESPEasy [22] nació en 2015 y marcó el inicio del firmware alternativo, demostrando la potencia y el control que era capaz de alcanzarse con los dispositivos y que hasta entonces estaban limitados a las funcionalidades que daba el fabricante. Actualmente está un poco desactualizado, pero sigue siendo una alternativa bastante sencilla para la programación de sensores y actuadores.

- ESPUrna

ESPUrna [23] es uno de los más completos ya que existe una imagen personalizada para cada dispositivo. Se utiliza principalmente para la programación de dispositivos Sonoff y leds y es muy utilizado ya que permite realizar flasheos directamente en los dispositivos sin tener que utilizar otras aplicaciones.

- ESPHome

ESPHome [24] es fácil de configurar mediante archivos YALM, permite la incorporación de nuevos dispositivos mediante plantillas y es bastante robusto. Ha permitido integrar automatizaciones y numerosos sensores debido a sus continuas actualizaciones.

- Tasmota

Tasmota [25] es un firmware de código abierto para dispositivos basados en ESP8266. Comenzó como una manera simple de piratear un Sonoff en un dispositivo controlado localmente, pero se ha convertido en un ecosistema completo para prácticamente cualquier dispositivo basado en ESP8266.

De las cuatro opciones, se ha elegido utilizar ESPHome ya que este programa permite crear un firmware personalizado para poder configurar y controlar dispositivos ESP32 simplemente configurando un fichero YAML. Además, permite la configuración de múltiples sensores en una tarjeta de desarrollo de manera más fácil e intuitiva.

5.2.3 Configuración de dispositivos

ESPHome [24] permite configurar diferentes proyectos de microcontrolador desde la web que lanza, pudiendo tener todos desde el mismo sitio e incluso pudiendo flashear directamente si se conecta el dispositivo directamente al equipo donde está instalado. En *Figura 51* se observa la interfaz web de ESPHome donde se podrán configurar los distintos dispositivos ESP32 de la red IoT de una manera sencilla e intuitiva. Para poder acceder a la interfaz web ha que conectarse al puerto 6065.

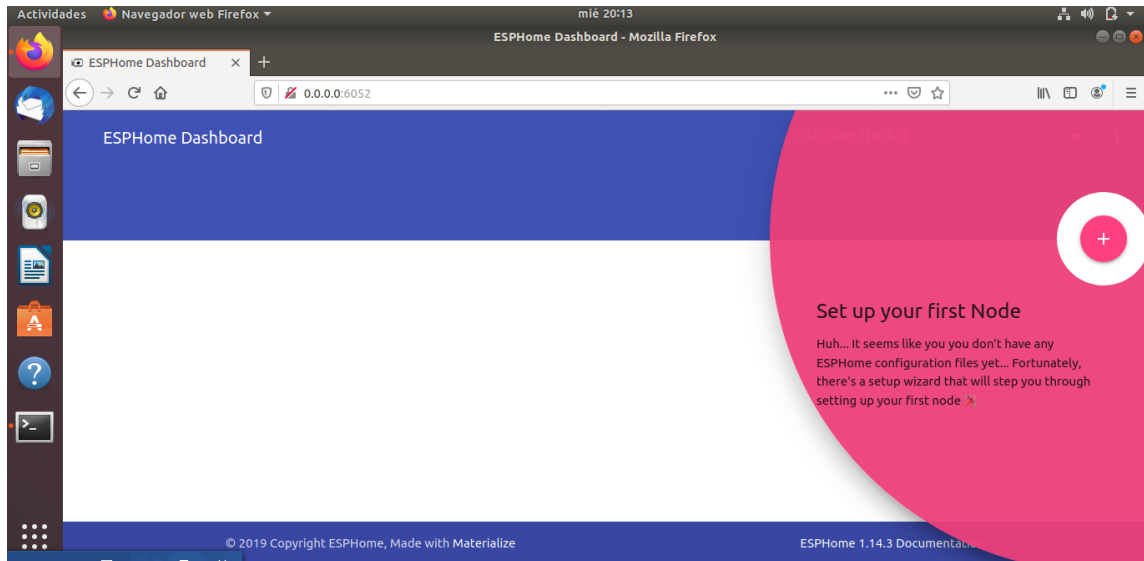


Figura 51:[Interfaz web ESPHome]

Para poder crear un nodo para cualquier dispositivo hay que seguir estos pasos:

1. Pulsar sobre el + que está en la esquina superior derecha, lo que abrirá una nueva pantalla que pide el nombre del dispositivo. Es posible poner el que queramos, pero es obligatorio usar solo caracteres en minúsculas y sin espacios ni caracteres raros.
2. Una vez puesto el nombre, pide el seleccionar el tipo de dispositivo sobre el que va a funcionar, que en nuestro caso es el ESP32, por lo que elegimos el genérico como aparece en Figura 52. Luego, pide los datos de la red wifi, así como una clave para poder actualizar el dispositivo por OTA.

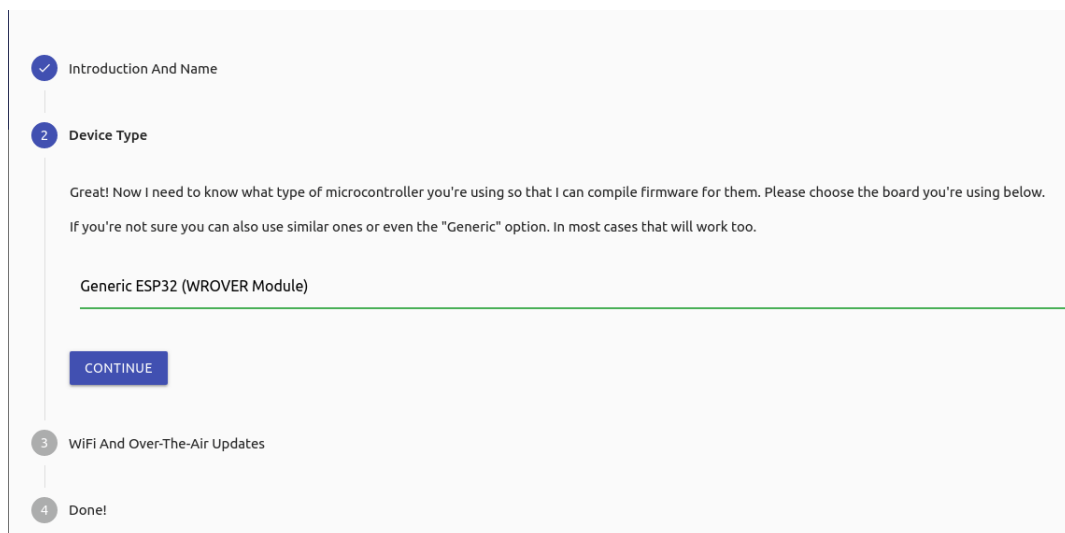


Figura 52:[Configuración de un nodo en ESPHome]

Una vez configurado el nodo, aparecerá en la interfaz web la instancia del dispositivo que acabamos de crear, así como, una opción “Edit” debajo del nombre para poder editar la configuración de dicho dispositivo. ESPHome ya dispone de una buena gama de componentes programados, pero también ofrece la posibilidad de crear sensores personalizados.



Figura 53:[Nodo ESP32]

5.2.4 Programación de los sensores

ESPHome tiene soporte para muchos sensores diferentes. Cada uno de ellos es una plataforma del dominio “sensor” y cada sensor tiene varias opciones de configuración base. Todos los sensores en ESPHome tienen un nombre y algunas otras opciones de configuración opcionales. La plataforma del sensor elegirá los valores apropiados para todos los sensores de forma predeterminada, pero siempre se pueden anular si se desea. En *Figura 54* se muestran algunos sensores disponibles en ESPHome.

Componentes del sensor









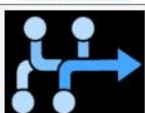


		ADE7953
Núcleo del sensor	ADC	ADE7953
		
ADS1115	AM2320	APDS9960
		
ATM90E32	AS3935	BH1750
		
Mapa de sensores binarios	BLE RSSI	BME280

Figura 54:[Sensores disponibles en ESPHome]

ESPHome también dispone de «customs components», dónde se podrá escribir el código C++ que se necesita para un componente que no se encuentra integrado en la aplicación, pero aprovechando el resto del sistema. Incluso existe la posibilidad en muchos casos de escribir código C++ directamente en el YALM del dispositivo con los llamados lambda, muy útil para cuando son fragmentos de código muy pequeños y sencillos.

5.2.5 ESPHome-Flasher

ESPHome-Flasher es una herramienta diseñada para flashear el firmware de dispositivos ESP32 a través de USB. Esta herramienta se puede usar para flashear los binarios del código programado con ESPHome y se empleará únicamente para realizar el primer flasheo, ya que posteriormente se podrá actualizar el firmware vía OTA.

En primer lugar, será necesario obtener el binario del firmware generado por ESPHome. Este puede conseguirse a través de la interfaz web, por lo que el proceso de flasheo se podrá realizar desde cualquier dispositivo conectado a la red. ESPHome-Flasher está disponible tanto para Windows como para Linux. Sin embargo, dado que, todos los ordenadores de la empresa tienen sistema operativo Windows hemos utilizado el programa específico para este sistema. Para realizar el primer flasheo será necesario seleccionar el puerto serie (en Windows son puertos COM) al que está conectado y el archivo BIN que queremos flashearle a dicho dispositivo.

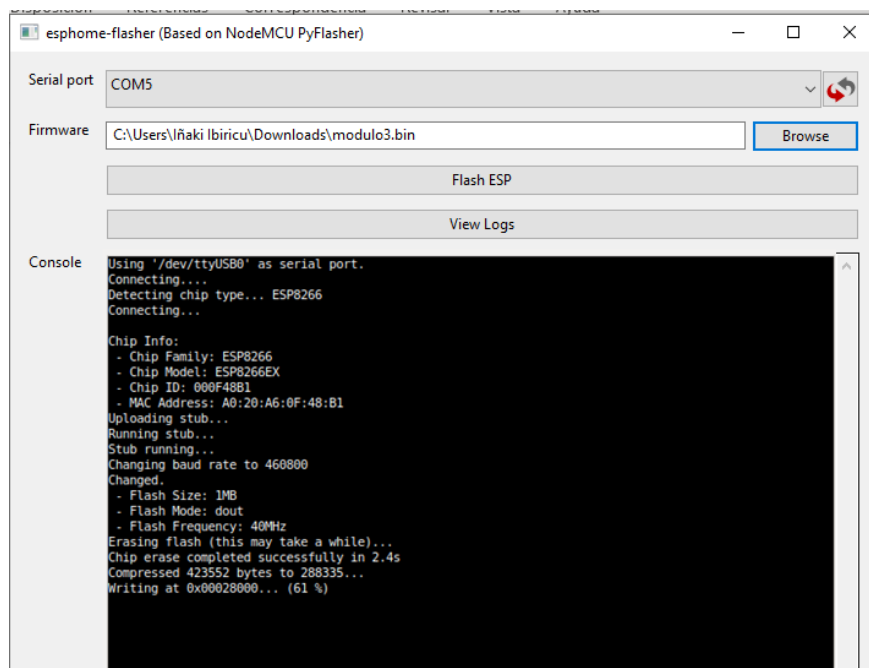
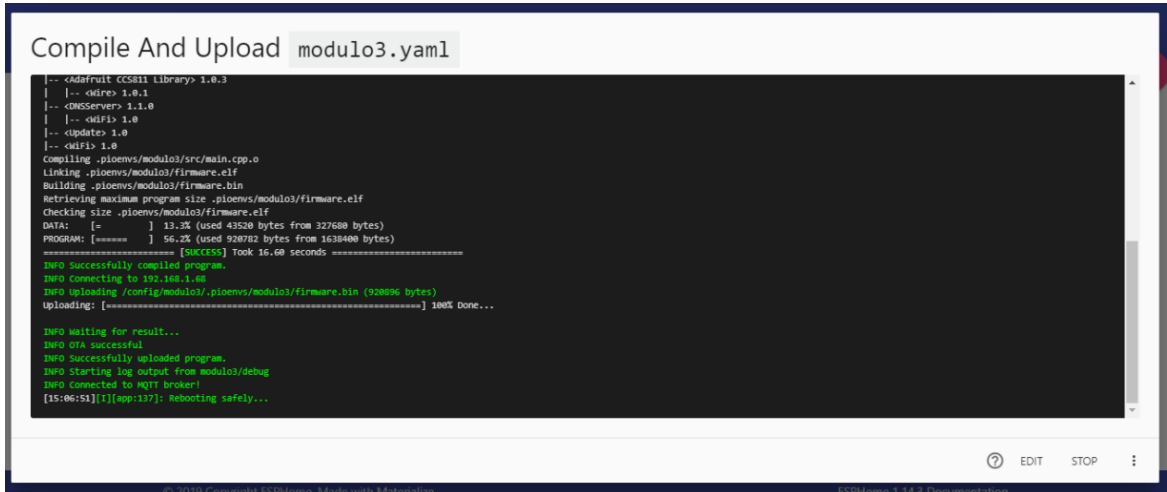


Figura 55:[Flasheo de dispositivo ESP32]

5.2.6 Actualización vía OTA (Over The Air)

Las siglas en inglés "OTA" se refieren a actualización por aire y es un método inalámbrico de provisión de un nuevo software o firmware a dispositivos. Las actualizaciones OTA son una manera muy efectiva de actualizar el software, solucionar errores y añadir o eliminar aplicaciones o cambiar la interfaz del usuario. Es imprescindible estar conectado a una red Wifi antes de poder usar la actualización OTA y mantener el dispositivo alimentado durante todo el proceso para evitar errores que podrían inutilizar el dispositivo. También es imprescindible que cada dispositivo tenga una dirección IP única, ya que la actualización del firmware se basa en buscar la dirección del dispositivo en la red y cargarle el nuevo firmware. Por eso es muy importante la labor del servidor DHCP. El proceso de actualización puede iniciarse directamente desde la interfaz web de ESPHome.



```

Compile And Upload modulo3.yaml
[--- <default> CCS811 Library 1.0.3
[--- <dire> 1.0.1
[--- <DMServer> 1.1.0
[--- <diFi> 1.0
[--- <update> 1.0
[--- <diFi> 1.0
Compiling .pioenvs/modulo3/src/main.cpp.o
Linking .pioenvs/modulo3/firmware.elf
Building .pioenvs/modulo3/firmware.bin
Retrieving maximum program size .pioenvs/modulo3/firmware.elf
Checking size .pioenvs/modulo3/firmware.elf
DATA: [=====] 13.2K (used 43520 bytes from 327680 bytes)
PROGRAM: [=====] 56.2K (used 928762 bytes from 1638400 bytes)
[=====] [SUCCESS] Took 16.68 seconds

DWM Successfully compiled program.
DWM Connecting to 192.168.1.68
DWM Uploading /config/modulo3/.pioenvs/modulo3/firmware.bin (928896 bytes)
uploading: [=====] 100% Done...

DWM waiting for result...
DWM OTA successful
DWM Successfully uploaded program.
DWM Starting log output from modulo3/debug
DWM Connected to MQTT broker!
[15:06:51][1][app:137]: rebooting safely...
  
```

Figura 56:[Actualización vía OTA]

5.3 Monitorización y almacenamientos de datos

5.3.1 MQTT

MQTT (Message Queue Telemetry Transport) es un protocolo de comunicación diseñado específicamente para la comunicación entre máquinas (M2M), por lo que posee características pensadas exclusivamente para este tipo de aplicaciones. Su modo de funcionamiento es a través de publicación/suscripción a un tópico, que se publica o lee. Sus mensajes son cortos, por lo que las transmisiones de datos también lo son y esto impacta en el consumo de energía de los nodos, algo crucial en dispositivos alimentados a baterías.

El protocolo MQTT consta de dos tipos de entidades en la red: un bróker de mensajería y numerosos clientes. El bróker es un servidor que recibe todos los mensajes de los clientes y los redirige a los clientes de destino relevantes. Por otra parte, un cliente es cualquier dispositivo que pueda interactuar con el bróker y enviar y recibir mensajes. Por tanto, un cliente puede ser desde un sensor de IoT en campo hasta una aplicación en un centro de datos que procesa datos de IoT.

Para que un cliente reciba un mensaje de un cliente tienen que pasar los siguientes acontecimientos:

1. El cliente se conecta al bróker. Puede suscribirse a cualquier "topic" de mensajería del bróker. Esta conexión puede ser una conexión TCP/IP simple o una conexión TLS cifrada para mensajes sensibles.
2. Otro cliente publica los mensajes en un "topic", enviando el mensaje y el "topic" al bróker.
3. Después, el bróker remite el mensaje a todos los clientes que se suscriben a este "topic".

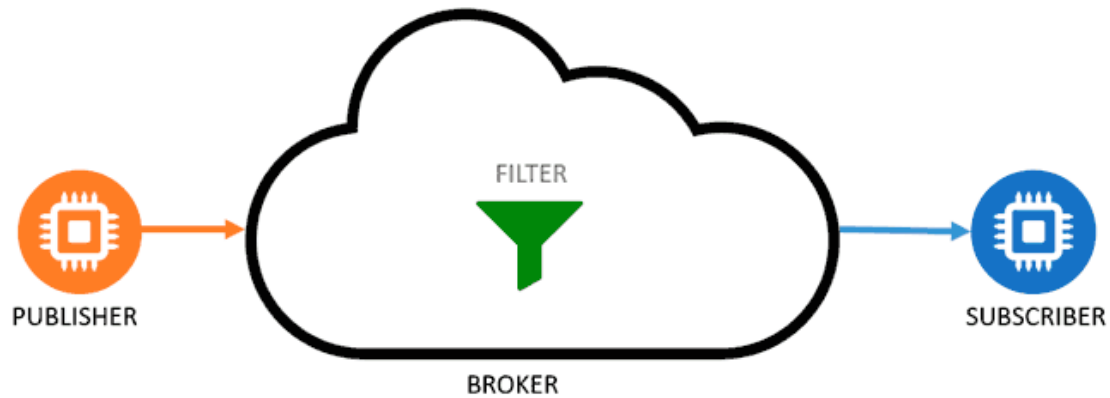


Figura 57:[Funcionamiento bróker MQTT]

MQTT dispone de un mecanismo de calidad del servicio o QoS, entendido como la forma de gestionar la robustez del envío de mensajes al cliente ante fallos (por ejemplo, de conectividad). MQTT tiene tres niveles QoS posibles:

- **QoS 0 unacknowledged (at most one):** El mensaje se envía una única vez. En caso de fallo el suscriptor puede que no reciba algún mensaje.
- **QoS 1 acknowledged (at least one):** El mensaje se envía hasta que se garantiza la entrega. En caso de fallo, el suscriptor puede recibir algún mensaje duplicado.
- **QoS 2 assured (exactly one):** Se garantiza que cada mensaje se entrega al suscriptor, y únicamente una vez.

Usar un nivel u otro depende de las características y necesidades de fiabilidad del sistema. Lógicamente, un nivel de QoS superior requiere un mayor intercambio de mensajes de verificación con el cliente y, por tanto, mayor carga al sistema y consumo energético. En este proyecto, se utilizará el protocolo QoS 0 ya que se envían mensajes cada poco tiempo y perder alguno no influye mucho.

En *Figura 58* podemos observar una pestaña de la línea de comandos donde se muestran los datos de los diferentes sensores del proyecto que están llegando al bróker MQTT.

```

[D][dallas.sensor:148]: 'Temp2ds18b20': Got Temperature=25.6°C
[D][sensor:092]: 'Temp2ds18b20': Sending state 25.56250 °C with 2 decimals of accuracy
25.56
[D][dht:048]: Got Temperature=25.6°C Humidity=63.6%
[D][sensor:092]: 'DHT22_Temp': Sending state 25.60000 °C with 2 decimals of accuracy
25.60
[D][sensor:092]: 'DHT22_Humidity': Sending state 63.60000 % with 0 decimals of accuracy
64
[D][adc:056]: 'MQ7_CO': Got voltage=0.76V
[D][sensor:092]: 'MQ7_CO': Sending state 0.75536 V with 2 decimals of accuracy
0.76
[D][adc:056]: 'MQ138_Formaldehido': Got voltage=0.92V
[D][sensor:092]: 'MQ138_Formaldehido': Sending state 0.91734 V with 2 decimals of accuracy
0.92
[D][dht:048]: Got Temperature=25.0°C Humidity=61.0%
[D][sensor:092]: 'DHT11_Temp': Sending state 25.00000 °C with 2 decimals of accuracy
25.00
[D][sensor:092]: 'DHT11_Humidity': Sending state 61.00000 % with 0 decimals of accuracy
61
[D][dallas.sensor:148]: 'Temp2ds18b20': Got Temperature=25.6°C
[D][sensor:092]: 'Temp2ds18b20': Sending state 25.56250 °C with 2 decimals of accuracy
25.56
[D][dht:048]: Got Temperature=25.5°C Humidity=63.5%
[D][sensor:092]: 'DHT22_Temp': Sending state 25.50000 °C with 2 decimals of accuracy
25.50
  
```

Figura 58:[Mensaje entrantes al bróker MQTT]

El protocolo MQTT dispone de distintas medidas de seguridad que podemos adoptar para proteger las comunicaciones. En muchos casos, la autenticación consiste en una contraseña y usuario que son enviados como texto plano, aunque también es posible configurar el bróker para aceptar conexiones anónimas. Para verificar la correcta configuración del bróker se pueden usar los comandos de publicación y suscripción. Por ejemplo, para verificar un el topic “test” con usuario y contraseña se podría usar:

```
mosquitto_sub -h <brokerhost> -t "test" -u "<username>" -P  
"<password>"  
mosquitto_pub -h <brokerhost> -t "test" -m "Hello World" -u  
"<username>" -P "<password>"
```

En este proyecto se va a emplear el bróker Eclipse Mosquitto. Mosquitto [26] que es un agente de mensajes de código abierto que implementa el protocolo MQTT, pero no requiere de muchos recursos para su ejecución. Esto hace que sea adecuado para usar en múltiples clases de dispositivos, desde ordenadores de una sola placa de baja potencia hasta servidores.

5.3.2 InfluxDB

InfluxDB [27] es una base de datos basada en series de tiempo de código abierto, lo que permite trabajar con ella de manera gratuita. Este tipo de bases de datos ha experimentado un crecimiento exponencial en los últimos años, con la popularización de IoT, Big Data y otras tecnologías que recogen gran cantidad de información en el tiempo. La información en InfluxDB está organizada en series de datos dentro de bases de datos. Dentro de cada base de datos encontramos al menos un campo, que representa la medida en sí misma (temperatura=20.9), aunque también podemos encontrar campos opcionales que contienen metadatos sobre el valor (lugar=Pamplona).

En este proyecto, InfluxDB está en el puerto 8086 y se ha creado una base de datos, a la que se le ha puesto el nombre de “rediot”, que contiene medidas de todos los dispositivos conectados a la red. Cada dispositivo guardará sus datos en una tabla o “measurement”, que contendrá información de todos los sensores conectados a dicho dispositivo como el valor, el topic, el tiempo, etc. En *Figura 59* podemos ver la información de una tabla del proyecto correspondiente a un dispositivo ESP32.

name: temperatura			
time	host	topic	value
----	----	-----	-----
1590862190997884713	73338880fd97	modulo3/sensor/mq138_formaldehido/state	0.91
1590862199837418260	73338880fd97	modulo3/sensor/dht11_temp/state	25
1590862199840583331	73338880fd97	modulo3/sensor/dht11_humidity/state	61
1590862204658730348	73338880fd97	modulo3/sensor/temp2ds18b20/state	25.63
1590862208675557512	73338880fd97	modulo3/sensor/mq7_co/state	0.74
1590862208675702572	73338880fd97	modulo3/sensor/dht22_temp/state	25.6
1590862208675776985	73338880fd97	modulo3/sensor/dht22_humidity/state	64
1590862251044326765	73338880fd97	modulo3/sensor/mq138_formaldehido/state	0.93
1590862259847384463	73338880fd97	modulo3/sensor/dht11_temp/state	25
1590862259847544319	73338880fd97	modulo3/sensor/dht11_humidity/state	61
1590862264643023650	73338880fd97	modulo3/sensor/temp2ds18b20/state	25.63
1590862268484772724	73338880fd97	modulo3/sensor/dht22_temp/state	25.6
1590862268492858780	73338880fd97	modulo3/sensor/dht22_humidity/state	64
1590862311137693812	73338880fd97	modulo3/sensor/mq138_formaldehido/state	0.91
1590862320049154138	73338880fd97	modulo3/sensor/dht11_temp/state	25
1590862320049340024	73338880fd97	modulo3/sensor/dht11_humidity/state	61
1590862325335752039	73338880fd97	modulo3/sensor/temp2ds18b20/state	25.63
1590862328493630308	73338880fd97	modulo3/sensor/dht22_temp/state	25.6
1590862328502051018	73338880fd97	modulo3/sensor/dht22_humidity/state	64
1590862358418512727	73338880fd97	modulo3/sensor/mq7_co/state	0.73
1590862371360812964	73338880fd97	modulo3/sensor/mq138_formaldehido/state	0.91
1590862379829333273	73338880fd97	modulo3/sensor/dht11_temp/state	25
1590862379829504289	73338880fd97	modulo3/sensor/dht11_humidity/state	61
1590862384640851404	73338880fd97	modulo3/sensor/temp2ds18b20/state	25.63

Figura 59:[Tabla de datos de InfluxDB]

Además, InfluxDB ofrece la opción de crear políticas de retención que definen por cuánto tiempo el motor mantiene los datos y cuántas copias se almacenan. Las políticas de retención son únicas por cada base de datos y junto con cada medición y tag definen una serie. Como se puede observar en *Figura 60*, se ha creado una política de retención llamada actualizar que almacena los datos de las últimas 4 semanas.

```
> show retention policies on sensores
```

name	duration	shardGroupDuration	replicaN	default
----	-----	-----	-----	-----
autogen	0s	168h0m0s	1	false
actualizar	720h0m0s	24h0m0s	1	true

Figura 60:[Política de retención de InfluxDB]

Para proteger la base de datos, se ha configurado un único usuario con permisos de administrador, es decir, que pueda añadir medidas, borrar datos, conceder permisos, etc. Además, se ha añadido otro usuario que solo tiene la posibilidad de leer una base de datos específica y que se usará para Grafana.

```
> show users
```

user	admin
----	-----
usertasu	true
grafana	false

Figura 61:[Usuarios de InfluxDB]

Hay que recordar que esta base de datos irá en otra máquina virtual dentro de un servidor junto con Telegraf como se puede ver en *Figura 6*. El objetivo de destinar una máquina virtual únicamente al almacenamiento de datos es para que el sistema vaya fluido y para que los recursos que se utilicen sean los mínimos posibles.

5.3.3 Telegraf

Telegraf [28] es un agente de servidor impulsado por complementos para recopilar y enviar métricas y eventos desde bases de datos, sistemas y sensores de IoT. Podemos encontrar 4 tipos de complementos distintos:

- **Complementos de entrada:** recopilan métricas del sistema, servicios o API de terceros.
- **Complementos del procesador:** transforman, decoran y / o filtran métricas.
- **Complementos de agregadores:** crean métricas agregadas (p. Ej., Promedio, mínimo, máximo, cuantiles, etc.).
- **Complementos de salida:** escriben métricas en varios destinos.

Dado que este servicio no requiere de acceso al exterior, el contenedor de Telegraf está enlazado directamente con el contenedor de InfluxDB, es decir, los datos que le transmite Telegraf a InfluxDB no llegan al exterior, sino que pasan a través del contenedor Docker0. En este proyecto, Telegraf se encargará de recopilar los datos que llegan al bróker de MQTT y guardar los datos en la base de datos. Para llegar a hacer esto, dispone de un archivo de configuración donde se configuran tanto las salidas como las entradas.

La configuración que usaremos en nuestro proyecto será la siguiente:

○ Entrada

Como entrada se ha elegido el consumidor de MQTT en el que hay que configurar principalmente la dirección IP de la máquina virtual donde está instalado el contenedor con el servidor MQTT y los topics (temas) a los que se va a suscribir Telegraf para leer datos. Dado que se desea almacenar toda la información recibida, se ha elegido como topic “#” para poder leer todos los mensajes que llegan al servidor MQTT. También hay que configurar otros parámetros como el nombre de la medida (name_override), el usuario y la contraseña del servidor MQTT y el formato de los datos que recibimos que en nuestro caso serán números decimales (float). A continuación, se muestra una plantilla de como quedaría la configuración:

```
[[input.MQTT_consumer]]
servers = ["tcp://IP_MQTT:1883"]
topics = ["#"]
name_override = "nombre_medida"
username = "usuario_MQTT"
password = "contraseña_MQTT"
data_format = "value"
data_type = "float"
```

○ Salida

Como salida se ha elegido, como es lógico, la base de datos InfluxDB. Para ello será necesario configurar la dirección donde se encuentra el contenedor donde está instalado InfluxDB que en nuestro caso será la dirección del localhost al estar Telegraf e InfluxDB instalados en la misma máquina virtual. Como datos opcionales se han configurado el nombre de la base de datos y el usuario y contraseña para que así Telegraf pueda escribir en ella. A continuación, se muestra una plantilla de como quedaría la configuración:

```
[[outputs.influxdb]]
urls = [http://127.0.0.1:8086]
database = "nombre_basededatos"
skip_database_creation = true
retention_policy = ""
write_consistency = "any"
timeout = "5s"
username = "ususario_admin"
password = "contraseña_admin"
```

5.4 Administración y visualización de datos

5.4.1 Grafana

Grafana [29] es una herramienta de software libre hecha para visualizar datos de series temporales de una manera fácil y agradable a través de una interfaz web. Esta información es recolectada y procesada por aplicaciones de terceros como pueden ser Cloudwatch, Graphite, Elasticsearch, OpenTSDB, Prometheus, Hosted Metrics o InfluxDB. Grafana nos ayuda a estudiar, analizar y monitorear datos durante un período de tiempo, permitiendo consultar, visualizar, configurar alertas y, en general, comprender los datos fácilmente con la ayuda de métricas.

El tablero de instrumentos está bastante equipado y evoluciona continuamente para dar sentido a los datos complejos. La herramienta tiene una gran cantidad de opciones de visualización para comprender los datos según nuestros requisitos comerciales, desde la visualización de gráficos hasta mapas de calor, histogramas, mapas geográficos.

En este proyecto, Grafana está instalada en el puerto 3000 y se usa principalmente para monitorizar todos los datos que se reciben de los sensores instalados en los dispositivos representándolos en gráficas y tablas. También se encarga de enviar alertas por correo electrónico en el caso en el que los valores de alguna medida sobrepasasen unos límites establecidos. En *Figura 62* se puede observar el "dashboard" de un módulo ESP32, donde se ven representados todos los datos de los sensores analizados anteriormente en tiempo real.

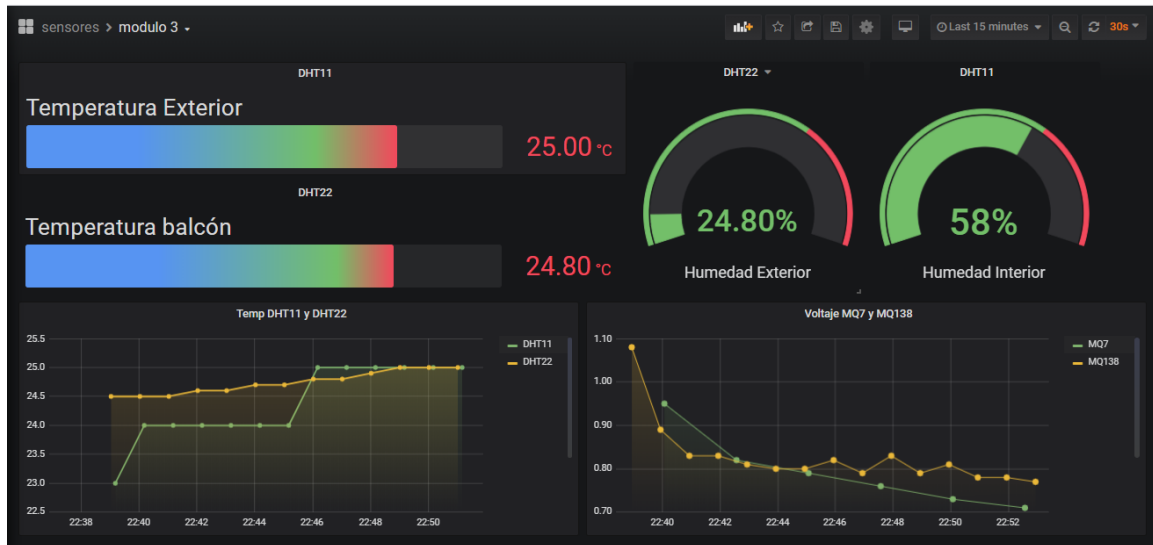


Figura 62:[Panel gráfico de Grafana]

5.4.2 Portainer

Portainer [30] es una herramienta web de código abierto que se ejecuta como un contenedor, por lo que requerirá disponer de una instalación de Docker. Esta aplicación nos va a permitir gestionar de forma muy fácil e intuitiva nuestros contenedores Docker a través de una interfaz web. Es útil para simplificar las tareas de administración de contenedores Docker, ya que, aunque es posible gestionarlos desde la línea de comandos, esto puede resultar a veces tedioso. Sin embargo, a través de una interfaz web, un administrador puede tener una visión global más clara de los contenedores que está ejecutando y facilitar su gestión.

Portainer nos va a permitir arrancar o parar un contenedor, cambiarles los nombres o incluso acceder a los logs de estos. La interfaz web está en el puerto 9000 y hace falta una autenticación de usuario para poder acceder a sus funciones.

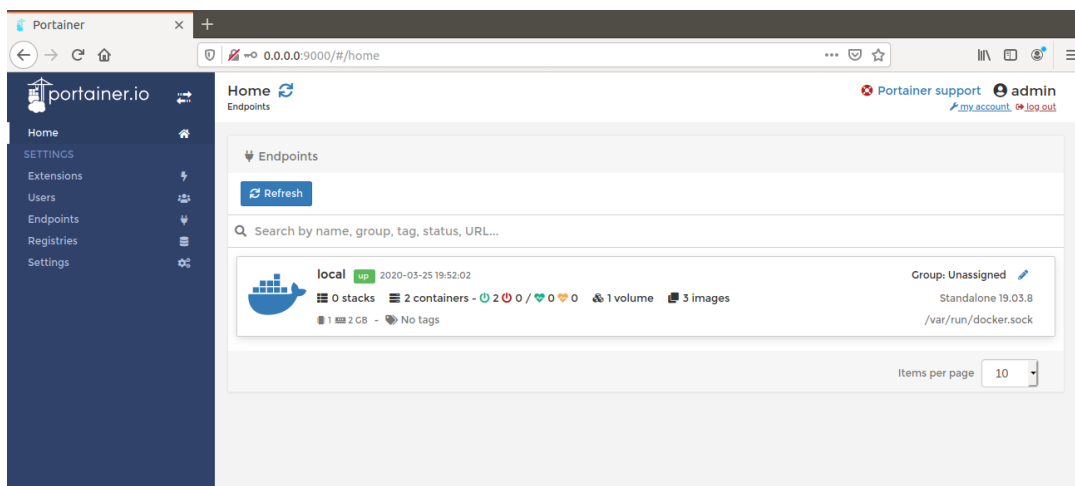


Figura 63:[Interfaz web Portainer]

5.4.3 Nginx

Nginx [31] es un servidor web de código abierto que, dado su éxito inicial, ahora también es usado como proxy inverso, cache de HTTP, y balanceador de carga. Entre sus usuarios más populares se encuentran Yahoo, Netflix, Cloudflare, WordPress.com, Adobe, Apple y muchos más. La estructura del software es asíncrona y controlada por eventos; lo cual permite el procesamiento de muchas solicitudes al mismo tiempo. Nginx también es altamente escalable, lo que significa que sus servicios aumentan a la par con el tráfico de sus clientes. El proceso de instalación de Nginx es bastante fácil, pues al haberse vuelto tan popular está presente en casi todos los repositorios de las principales distribuciones Linux más populares de hoy en día, incluso para Docker.

En este proyecto utilizaremos Nginx para albergar un portal web que nos permitirá acceder e interaccionar con las páginas web de todas las aplicaciones del proyecto tales como Grafana, ESPHome, el servidor DHCP y Portainer. La página web está en el puerto 80, que se conoce como puerto por default por el cual un servidor HTTP “escucha” la petición hecha por un cliente, es decir por una PC en específico.

Para el desarrollo del portal web se optó por el uso del programa Visual Studio Code puesto que aparte de ser gratuito y de código abierto, es también ágil, rápido y potente. En lo que se refiere a HTML y CSS ofrece grandes características como, por supuesto, la completitud automática de código, ayuda contextual a medida que escribes, selectores de colores, refactorización de código y depuración sin salir de la herramienta.

El portal desarrollado (Figura 64) cuenta con una página de inicio donde se encuentra una breve introducción de todos los programas que la integran, así como una explicación de cómo funcionan dentro del proyecto. También existe un apartado de navegación donde se puede acceder a los programas por medio de iframes por lo que se podrá interactuar en ellas sin tener que salir del portal web. La página cuenta con un título y un pie de página donde se encuentra información acerca de la empresa.

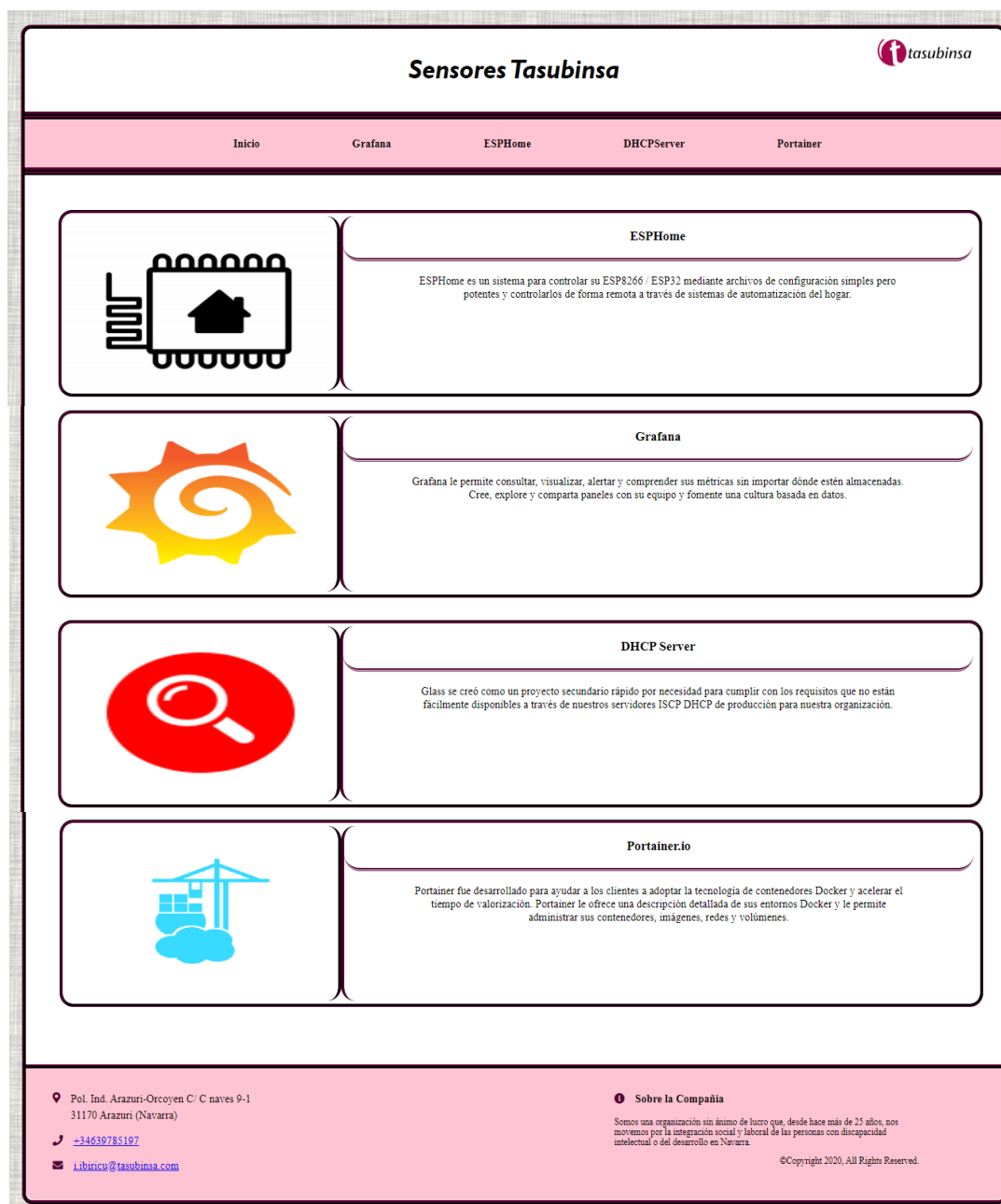


Figura 64:[Página web]

- Autenticación de usuario

Para evitar accesos no autorizados al portal, se configuró el acceso limitado empleando usuario y contraseña mediante un archivo HTPASS. Esto aporta seguridad al portal web ya que impedirá el acceso de personal no autorizado a la misma.

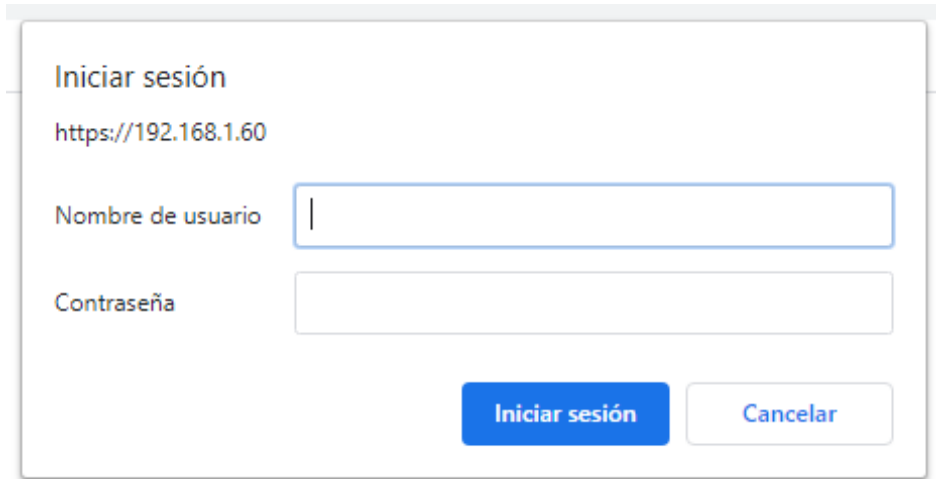
El formulario de inicio de sesión tiene un título "Iniciar sesión" en azul. Debajo del título, se muestra la URL "https://192.168.1.60". Hay dos campos de entrada: "Nombre de usuario" con un cursor de texto visible y "Contraseña". En la parte inferior derecha, hay dos botones: "Iniciar sesión" en azul y "Cancelar" en gris.

Figura 65:[Autenticación de usuario]

- Certificados SSL

SSL es el acrónimo de Secure Sockets Layer, la tecnología estándar para mantener segura una conexión a Internet, así como para proteger cualquier información confidencial que se envía entre dos sistemas. En este proyecto se usarán certificados SSL para garantizar la seguridad en las comunicaciones, impidiendo que cualquier persona pueda leer y modificar cualquier dato que se transfiera, incluida información que pudiera considerarse personal.

HTTPS (protocolo seguro de transferencia de hipertexto) aparece en la dirección URL cuando un sitio web está protegido por un certificado SSL. Hay varias formas de conseguir certificados SSL, existen de pago, gratuitos y también se pueden hacer certificados personalizados. En este proyecto se ha optado por utilizar la herramienta OpenSSL [32] para crear el certificado autofirmado. Gracias a que son personalizados, los detalles del certificado, como, por ejemplo, la entidad emisora y el nombre corporativo del propietario del sitio web, se pueden personalizar.

6. Problemas

En un primer lugar el proyecto estaba pensado para desarrollarse dentro de las instalaciones de Tasubinsa, ya que el objetivo era montar todo el sistema en los servidores de una de las plantas de Tasubinsa y probar el funcionamiento de este en distintos procesos de producción para comprobar el rendimiento y la fiabilidad. Pero, debido al coronavirus, se decidió simular el comportamiento de la red IoT montando todo el sistema en un ordenador portátil que se asemeja lo máximo posible a lo que sería un servidor de una empresa. Además, no se ha podido configurar el servidor DHCP ya que, en caso de hacerlo, el servidor generaría conflicto con el router instalado en casa.

Otro de los grandes problemas que ha generado el coronavirus está relacionado con los sensores. Algunos sensores como el de conductividad y el de control de pH estaban pedidos, pero estos pedidos se cancelaron por motivos de seguridad. Por otra parte, otros sensores como los MQ no se recibieron hasta finales de mayo por lo que no ha sido posible dedicarles el tiempo suficiente para realizar una conversión y una calibración efectiva.

Pero, dejando a un lado los problemas causados por el coronavirus, durante el desarrollo del proyecto han aparecido algunos inconvenientes que por motivos de tiempo no se han podido solucionar. Los más importantes son los relacionados con la página web:

- Al desarrollar la programación de la página web se decidió usar iframes para la visualización e interacción de las distintas aplicaciones que componen el proyecto. El problema principal de esto es que la página web utiliza el protocolo HTTPS mientras que las interfaces web de las aplicaciones del proyecto son HTTP. Esto hace que, para que los iframes funcionen, sea necesario deshabilitar la función de contenido no seguro dentro del navegador. Este problema se puede solucionar utilizando un proxy SSL que redirige todas las páginas a HTTPS.
- El sistema de autenticación utilizado para proteger la página web funciona correctamente pero no es muy eficaz si se desea implementar en una página web debido a que solo pide el usuario y la contraseña y eso hace que la seguridad sea muy pobre. Para solucionar esto se suele utilizar lo que se conoce como doble autenticación que consiste en añadir otro factor más para poder acceder a la página. Este factor extra suele ser un token físico como puede ser un móvil al que únicamente tiene acceso el usuario autorizado.

7. Conclusiones y líneas futuras

La primera conclusión que se puede extraer de este proyecto es que no se han podido cumplir todos los objetivos del mismo. El proyecto inicialmente era muy ambicioso y era realizable en el plazo de tiempo establecido, pero la presencia del coronavirus ha hecho que no se haya podido implantar completamente, ya que no se ha podido volcar el sistema en un servidor físico. Sin embargo, a pesar de las dificultades se ha conseguido desarrollar un sistema IoT de monitorización y realizar una maqueta del mismo plenamente funcional. Esta maqueta dispone de una interfaz web unificada que hace que este proyecto sea fácil de usar para cualquier usuario sin conocimientos sobre el tema pese a la dificultad interna que este supone.

Por otra parte, es destacable el uso de máquinas virtuales para implementar todos los servicios de la red. El haber implementado los huéspedes que alojan los contenedores Docker en máquinas virtuales va a permitir su rápida implantación en los servidores de la empresa, ya que únicamente será necesario copiar las mismas y modificar mínimamente su configuración. Por otra parte, el uso de Docker como sistema de virtualización de servicios proporciona a este proyecto de una gran escalabilidad. Además, en caso de no querer usarse las máquinas virtuales desarrolladas o querer cambiarse los servicios ubicados en las mismas, será posible cambiarlos de máquina al ser muy sencillo trasladar contenedores Docker entre sistemas.

Finalmente cabe destacar que este proyecto ha supuesto un reto ya que los conocimientos necesarios para el desarrollo del proyecto son bastante amplios al englobar nociones de electrónica para la configuración de los sensores, de telecomunicaciones para establecer toda la red de comunicación entre los dispositivos y el servidor e incluso de informática para el desarrollo de la página web.

En cuanto a las líneas futuras de trabajo, como ya se ha comentado han quedado unas cuantas tareas pendientes, entre las que se incluyen:

- El desarrollo del firmware para la lectura de todos los sensores, ya que al no recibirse los mismos no fue posible programarlos y comprobar su funcionamiento.
- La implantación de los servicios en la red informática de Tasubinsa y verificación de funcionamiento.

Además de estas tareas que se han quedado pendientes, a este proyecto se le pueden implementar bastantes mejoras entre las que destacan tres que supondría un avance en cuanto a comodidad y protección.

- La mejora del portal web solucionando los problemas con los iFrames e implantando la autenticación en dos pasos.
- Desarrollar una carcasa con tecnología 3D para la protección de todas las conexiones de los sensores con el dispositivo, así como mejoraría en cuanto a apariencia y transporte.
- Desarrollar un sistema de baterías que faciliten la colocación de los dispositivos ESP32 en cualquier lugar sin la necesidad de tener cerca una fuente de alimentación de red.

8. Bibliografía

- [1] M2M. URL: <https://smart-it.io/machine-to-machine-m2m-how-does-it-work/>.
- [2] (TENG), «Triboelectric Nanogenerator,» URL: <https://phys.org/news/2019-04-next-generation-triboelectric-nanogenerator-teng-constant.html>.
- [3] (IEEE), «Instituto de Ingeniería Eléctrica y Electrónica,» URL: <https://www.ieee.org/>.
- [4] MQTT. URL: <http://mqtt.org/>.
- [5] Espressif, «Microcontrolador ESP8266,» URL: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.
- [6] Datasheet, «Dispositivo ESP01,» URL: <http://www.microchip.ua/wireless/esp01.pdf>.
- [7] Espressif, «Microcontrolador ESP-WROOM-32,» URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.
- [8] Espressif, «Dispositivo ESP32,» URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.
- [9] Datasheet, «Dispositivo ESP32 Node MCU,» URL: http://wiki.ai-thinker.com/_media/esp32/docs/nodemcu-32s_product_specification.pdf.
- [10] Datasheet, «Sensor DHT11,» URL: <http://www.micro4you.com/files/sensor/DHT11.pdf>.
- [11] Datasheet, «Sensor DHT22,» URL: <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>.
- [12] Datasheet, «Sensor CCS811,» URL: https://cdn.sparkfun.com/assets/learn_tutorials/1/4/3/CCS811_Datasheet-DS000459.pdf.
- [13] Datasheet, «Sensor MQ7,» URL: <https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf>.
- [14] Datasheet, «Sensor MQ138,» URL: <http://www.china-total.com/Product/meter/gas-sensor/MQ138.pdf>.
- [15] Datasheet, «Sensor DS18B20,» URL: <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>.
- [16] DFROBOT, «Sensor DFR0300,» URL: <https://uk.farnell.com/dfrobot/dfr0300/analog-electrical-conductivity/dp/2946108>.
- [17] DFROBOT, «Sensor pH SEN0161,» URL: https://www.openhacks.com/uploadsproductos/ph_meter_sku__sen0161_-_robot_wiki.pdf.
- [18] VMWare_Workstation. URL: <https://www.vmware.com/es.html>.
- [19] Docker. URL: <https://www.docker.com/>.
- [20] Docker_Hub. URL: <https://hub.docker.com/>.
- [21] Ionos, «Servidor DHCP,» URL: : <https://www.ionos.es/digitalguide/servidores/configuracion/que-es-el-dhcp-y-como-funciona/>.
- [22] ESPEasy. URL: <https://www.letscontrolit.com/wiki/index.php/ESPEasy>.
- [23] ESPurna. URL: <https://www.esputna.org/>.
- [24] ESPHome. URL: <https://esphome.io/>.
- [25] Tasmota. URL: <https://tasmota.github.io/docs/>.
- [26] Mosquitto. URL: <https://mosquitto.org/>.
- [27] InfluxDB. URL: <https://www.influxdata.com/>.
- [28] Telegraf. URL: <https://www.influxdata.com/time-series-platform/telegraf/>.
- [29] Grafana. URL: <https://grafana.com/>.

[30] Portainer. URL: <https://www.portainer.io/>.

[31] Nginx. URL: <https://www.nginx.com/>.

[32] OpenSSL. URL: <https://www.openssl.org/>.

A. Código del proyecto en Linux (Ubuntu)

A.1 Instalar Docker en Ubuntu

Este es el código para instalar Docker en Ubuntu y que va a permitir crear contenedores para el resto de las aplicaciones.

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
sudo apt update
sudo apt-cache policy docker-ce
sudo apt install docker-ce
```

Para comprobar si Docker funciona correctamente se emplea este comando.

```
sudo systemctl status docker
<Ctrl+C>
```

Estos son algunos comandos necesarios para poder administrar los contenedores que se instalen en el sistema.

```
sudo docker container ls -a
sudo docker start <nombre contenedor>
sudo docker stop <nombre contenedor>
sudo docker rm <nombre contenedor>
sudo docker inspect <nombre contenedor>
```

A.2 Instalar Portainer.io en un contenedor

Para instalar la aplicación de Portainer en un contenedor, emplearemos este código donde aparte de asignar un puerto exterior, se configurará para que se reinicie en caso de fallo.

```
sudo apt update
sudo docker volume create portainer_data
sudo docker run -d --name=portainer --restart=always -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer
```

Al iniciar por primera vez la aplicación web de Portainer, se pedirá un usuario y una contraseña, que son los que se muestran en el código. A continuación, la propia aplicación te da la opción de poner una contraseña nueva.

```
Usuario → admin
Contraseña → admin
-----
Nueva contraseña → <contraseña>
```

A.3 Instalar MQTT en un contenedor

Estos son los comandos que hay que utilizar para descargar la imagen Docker con el servidor MQTT de Eclipse-Mosquitto.

```
sudo apt update
sudo docker pull eclipse-mosquitto
```

Con estos códigos se crearán carpetas para poder acceder a las distintas configuraciones que ofrece el servidor MQTT sin tener que entrar en el contenedor.

```
sudo mkdir -p /proyecto/mosquitto/config
sudo mkdir -p /proyecto/mosquitto/data
sudo mkdir -p /proyecto/mosquitto/log
```

Con esta parte del código crearemos en la carpeta de configuración del servidor MQTT una plantilla de configuración para Eclipse-Mosquitto.

```
cd
sudo git clone https://github.com/eclipse/mosquitto.git eclipse-
mosquitto
cd eclipse-mosquitto/
sudo cp mosquitto.conf /proyecto/mosquitto/config/
cd
```

Con este comando se creará el contenedor con el bróker Eclipse-Mosquitto donde se le asignarán los puertos exteriores, los volúmenes y para que se reinicie en caso de fallo.

```
sudo docker run -it \
--name=MQTT \
--restart=always \
-p 1883:1883 -p 9001:9001 \
-v /proyecto/mosquitto/config:/mosquitto/config \
-v /proyecto/mosquitto/data:/mosquitto/data \
-v /proyecto/mosquitto/log:/mosquitto/log \
eclipse-mosquitto

<Ctrl+C>
```

Para probar el funcionamiento del bróker MQTT hay que descargar una aplicación donde se podrán publicar mensajes y subscribirse a cualquier tema en la dirección en la que se encuentra el servidor.

```
sudo apt install mosquitto-clients

sudo mosquitto_sub -h <dirección MQTT> -t "topic"
sudo mosquitto_pub -h <dirección MQTT> -t "topic" -m "mensaje"
```

Una forma de agregar seguridad al servidor MQTT es añadiendo en el contenedor un archivo de contraseña con estos comandos.

```
sudo docker exec -it MQTT sh
mosquitto_passwd -c /mosquitto/config/pwfile <usuario>
## password ## → <contraseña>
## reenter password ## → <contraseña>

exit
```

Para añadir más usuarios se utilizarán los siguientes comandos.

```
sudo docker exec -it MQTT sh  
mosquitto_passwd -b /mosquitto/config/pwfile <usuario> <contraseña>
```

Una forma de ver todos los usuarios del servidor MQTT es mediante el siguiente comando.

```
sudo docker exec -it MQTT sh  
cat /mosquitto/config/pwfile
```

Una vez añadido un archivo de contraseña hay que cambiar la configuración del archivo mosquitto.conf.

```
cd /proyecto/mosquitto/config  
sudo nano mosquitto.conf
```

Hay que añadir esto al archivo mosquitto.conf donde se bloquea el acceso a usuarios anónimos y se definen la dirección donde se encuentra el archivo de contraseña.

```
allow_anonymous false  
password_file /mosquitto/config/pwfile
```

Para que los cambios surjan efecto, será necesario reiniciar el contenedor.

```
sudo docker container restart MQTT
```

Después de reiniciar el contenedor, los comandos de publicación y suscripción deberán tener un nombre de usuario y contraseña para funcionar.

```
mosquitto_sub -h <dirección MQTT> -t "topic" -u "<usuario>" -P  
"<contraseña>"  
mosquitto_pub -h <dirección MQTT> -t "topic" -m "mensaje" -u  
"<usuario>" -P "<contraseña>"
```

A.4 Instalar ESPHome en un contenedor

Para crear un contenedor con ESPHome hay que utilizar estos comandos donde se define un puerto exterior y se crea una carpeta para tener accesible la configuración de la aplicación.

```
sudo apt update  
sudo docker pull esphome/esphome  
sudo mkdir -p /proyecto/esphome  
  
sudo docker run -it --name=esphome -p 6052:6052 \  
-v /proyecto/esphome:/config -d esphome/esphome
```

Para crear un sensor personalizado hay que crear un archivo con extensión .H en la carpeta de configuración de ESPHome con el código del sensor utilizando este comando.

```
sudo gedit /proyecto/esphome/sensor_personalizado.h
```

A.5 Instalar InfluxDB en un contenedor

Para crear un contenedor con InfluxDB, hay que utilizar estos comandos donde se crean las carpetas necesarias para poder acceder a la configuración de este y además se expone un puerto al exterior.

```
sudo apt update
sudo docker pull influxdb

sudo useradd -rs /bin/false influxdb
sudo mkdir -p /proyecto/influxdb
sudo docker run --rm influxdb influxd config | sudo tee
/proyecto/influxdb/influxdb.conf > /dev/null
sudo chown influxdb:influxdb /proyecto/influxdb/
sudo mkdir -p /var/lib/influxdb
sudo chown influxdb:influxdb /var/lib/influxdb

sudo docker run -d -p 8086:8086 -p 8083:8083 --name=influxdb \
-v /proyecto/influxdb/influxdb.conf:/etc/influxdb/influxdb.conf \
-v /var/lib/influxdb:/var/lib/influxdb \
influxdb \
-config /etc/influxdb/influxdb.conf
```

InfluxDB permite crear bases de datos y ver las medidas que hay almacenadas a través de los siguientes comandos.

```
sudo docker exec -it influxdb influx
show databases
create database <nombre base de datos>
use <nombre base de datos>
show measurements
select * from <nombre medida>
```

A continuación, hay una lista de comandos que admite InfluxDB para realizar diferentes funciones.

```
Crear bases de datos → create database <nombreddb>
Borrar bases de datos → drop database <nombreddb>
Mostrar políticas de retención → show retention policies on <nombreddb>
Crear políticas de retención → create retention policy "nombrepr" on
"nombreddb" duration 30d replication 1 default
Borrar políticas de retención → drop retention policy "nombrepr" on
"nombreddb"
Mostrar consultas continuas → show continuous queries
Crear consultas continuas → create continuous query "nombrecc" on
"nombreddb" begin select min("field") into "target_measurement" from
"current_measurement" group by time(30m) end
Borrar consultas continuas → drop continuous query "nombrecc" on
"nombreddb"
Mostrar usuarios → show users
Borrar usuario → drop user "username"
Mostrar estadísticas → show stats
Mostrar diagnósticos → show diagnostics
```

Para crear un administrador con todos los privilegios para crear y editar cualquier base de datos hay que utilizar estos comandos.

```
sudo docker exec -it influxdb influx
create user <usuario admin> with password '<contraseña admin>' with
all privileges
exit
```

Hay que cambiar el archivo de configuración influxdb.conf después de añadir un administrador.

```
sudo nano /proyecto/influxdb/influxdb.conf
```


En el archivo hay que cambiar el valor de la variable auth-enabled por "true".

```
[http]
enabled = true
bind-address = ":8086"
auth-enabled = true
```

Para que los cambios surjan efecto hay que reiniciar el contenedor.

```
sudo docker container restart influxdb
```

Con los siguientes comandos se puede crear un usuario que solo tenga acceso a la lectura de medidas de una base de datos de InfluxDB. Se utilizará un usuario con estas características para Grafana.

```
sudo docker exec -it influxdb influx -username <usuario admin> -
password 'contraseña admin'
create user <usuario> with password 'contraseña'
create database <nombre base de datos>
grant read on "nombre base de datos" to "usuario"
exit
```

A.6 Instalar Telegraf en un contenedor

Estos comandos se utilizan para descargar la imagen de Telegraf para Docker y para crear las carpetas para poder acceder a la configuración de esta.

```
sudo apt update
sudo docker pull telegraf

sudo useradd -rs /bin/false telegraf
sudo mkdir -p /proyecto/telegraf
sudo docker run --rm telegraf telegraf config | sudo tee
/proyecto/telegraf/telegraf.conf > /dev/null
sudo chown telegraf:telegraf /proyecto/telegraf/*
```

Hay que cambiar el archivo de configuración telegraf.conf para determinar las entradas y salidas.

```
cd /proyecto/telegraf/
sudo nano telegraf.conf
```

donde habrá que cambiar las siguientes líneas de la configuración General, Input y Output.

```
[General]
interval = "10s"
round_interval = true
metric_batch_size = 1000
metric_buffer_limit = 10000
collection_jitter = "0s"
flush_interval = "10s"
flush_jitter = "0s"
precision = ""
hostname = ""
omit_hostname = false

[Output]
[[outputs.influxdb]]
urls = [http://127.0.0.1:8086]
database = "nombre basededatos"
skip_database_creation = true
```

```
retention_policy = ""
write_consistency = "any"
timeout = "5s"
username = "ususuario_admin"
password = "contraseña_admin"
[Input]
< Hay valores que vienen por defecto como cpu, disk, memory, ... >
[[input.MQTT_consumer]]
servers = ["tcp://IP_MQTT:1883"]
topics = ["#"] ~ Con este topic accedes a todos los mensajes
               que llegan al bróker ~
name_override = "nombre_medida"
username = "usuario_MQTT"
password = "contraseña_MQTT"
data_format = "value"
data_type = "float"
```

Con la configuración adecuada se puede crear el contenedor de Telegraf con el siguiente comando.

```
sudo docker run -d --name=telegraf \
  --net=container:influxdb \
  -e HOST_PROC=/host/proc \
  -v /proc:/host/proc:ro \
  -v /proyecto/telegraf/telegraf.conf:/etc/telegraf/telegraf.conf
\
  telegraf
```

Este comando se utiliza para ver los logs de telegraf.

```
sudo docker container logs -f telegraf
```

A.7 Instalar DHCP en un contenedor

Para crear un contenedor con un servidor DHCP se utiliza este código donde se establecen los puertos exteriores y el usuario y la contraseña de la aplicación web.

```
sudo apt update
sudo docker pull djaydev/glass-isc-dhcp

sudo docker run -d \
  --restart=always \
  --name=dhcpserver \
  --network="host" \
  -e TZ=Europa/Madrid \
  -v /proyecto/dhcp:/etc/dhcp:rw \
  -v /proyecto/dhcp/leases:/var/lib/dhcp:rw \
  -e ADMINUSER=<usuario> \
  -e ADMINPASSWORD=<contraseña> \
  -e WEBSOCKETPORT=8080 \
  -e WEBADMINPORT=3001 \
  djaydev/glass-isc-dhcp
```

Hay que crear un archivo de configuración al servidor DHCP para que funcione correctamente.

```
sudo gedit /proyecto/dhcp/dhcpd.conf
```

Para modificar la configuración dentro de la aplicación web hay que introducir el usuario y contraseña establecidos al crear el contenedor.

```
Usuario → <Usuario>
Contraseña → <contraseña>
```

A.8 Instalar Grafana en un contenedor

Para crear un contenedor con Grafana hay que utilizar este código donde se establece el puerto exterior y se habilita la opción de poder cargar Grafana en iframes.

```
sudo apt update
sudo docker pull grafana/grafana
sudo docker volume create grafana-storage
sudo docker run -d -p 3000:3000 --name=grafana \
  -e GF_SECURITY_ALLOW_EMBEDDING=true \
  -v grafana-storage:/var/lib/grafana grafana/grafana
```

Al iniciar por primera vez la aplicación web de Grafana, te pedirán un usuario y una contraseña, que son los que se muestran en el código. A continuación, la propia aplicación dará la opción de poner una contraseña nueva.

```
Usuario → admin
Contraseña → admin
-----
Nueva Contraseña → <contraseña>
```

A.9 Instalar Nginx en un contenedor

Antes de crear el contenedor con Nginx hay que crear una página web personalizada para el servidor web. Con los comandos que se muestran a continuación, se crea un directorio que alojará contenido del sitio web.

```
sudo docker pull nginx
sudo mkdir -p /proyecto/nginx/html
cd /proyecto/nginx/html
```

En esta carpeta hay que incluir los archivos formato.css y index.html de la página de inicio.

```
sudo mkdir -p /proyecto/nginx/html/inicio
cd /proyecto/nginx/html/inicio
```

En esta carpeta hay que incluir los archivos formato.css y index.html de la página de Grafana.

```
sudo mkdir -p /proyecto/nginx/html/grafana
cd /proyecto/nginx/html/grafana
```

En esta carpeta hay que incluir los archivos formato.css y index.html de la página de ESPHome.

```
sudo mkdir -p /proyecto/nginx/html/esphome
cd /proyecto/nginx/html/esphome
```

En esta carpeta hay que incluir los archivos formato.css y index.html de la página del servidor DHCP.

```
sudo mkdir -p /proyecto/nginx/html/dhcp
cd /proyecto/nginx/html/dhcp
```

En esta carpeta hay que incluir los archivos formato.css y index.html de la página de Portainer.

```
sudo mkdir -p /proyecto/nginx/html/portainer
cd /proyecto/nginx/html/portainer
```

Se crea el contenedor con Nginx con el siguiente comando

```
sudo docker run --name=nginx --restart=always \
-p 80:80 -v /proyecto/nginx/html:/usr/share/nginx/html nginx
```

Una vez creado el contenedor se copia la configuración de Nginx y se destruye el contenedor con la finalidad de añadir seguridad y certificados SSL.

```
cd /proyecto/nginx
sudo docker cp /proyecto/nginx:/etc/nginx/conf.d/default.conf
default.conf
sudo docker stop nginx
sudo docker rm nginx
```

Una forma de agregar seguridad al servidor de Nginx es añadiendo en el contenedor un archivo de contraseña con estos comandos.

```
sudo apt-get update
sudo apt-get install apache2-utils

sudo htpasswd -c /proyecto/nginx/.htpasswd usuario
## password ## → <contraseña>
## reenter password ## → <contraseña>
```

Para añadir más usuarios se tienen que utilizar los siguientes comandos.

```
sudo htpasswd /proyecto/nginx/.htpasswd usuario
## password ## → <contraseña>
## reenter password ## → <contraseña>
```

Para ver los usuarios se utiliza el siguiente comando.

```
cat /proyecto/nginx/.htpasswd
```

Los certificados SSL para conseguir una conexión https se consiguen mediante una aplicación llamada OpenSSL. Utilizando el siguiente código se crearán certificados totalmente personalizados.

```
sudo apt install openssl
cd /proyecto/nginx
mkdir SSL
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/proyecto/nginx/SSL/nginx.key -out /proyecto/nginx/SSL/nginx.crt
```

Es necesario modificar la plantilla de configuración de Nginx que se encuentra en el archivo default.conf para que añada estos cambios.

```
cd /proyecto/nginx
sudo nano default.conf
```

Hay que añadir este archivo de configuración donde ya están implementados los certificados SSL, la autorización de usuario y se habilitan los iframes en la página web.

```
server {
    listen      80;
    rewrite ^ https://$http_host/;
}

server {
    listen 443 SSL;

    SSL on;
    SSL_certificate /etc/nginx/SSL/nginx.crt;
    SSL_certificate_key /etc/nginx/SSL/nginx.key;

    add_header X-Frame-Options "SAMEORIGIN";

    #charset koi8-r;
    access_log /var/log/nginx/host.access.log main;

    auth_basic "Administrator Login";
    auth_basic_user_file /etc/nginx/.htpasswd;

    root /usr/share/nginx/html/index;
    index index.html index.htm;

    location /grafana/ {
        alias /usr/share/nginx/html/grafana/;
        add_header X-Frame-Options "SAMEORIGIN";
    }

    location /esphome/ {
        alias /usr/share/nginx/html/esphome/;
        add_header X-Frame-Options "SAMEORIGIN";
    }

    location /dhcp/ {
        alias /usr/share/nginx/html/dhcp/;
        add_header X-Frame-Options "SAMEORIGIN";
    }

    location /portainer/ {
        alias /usr/share/nginx/html/portainer/;
        add_header X-Frame-Options "SAMEORIGIN";
    }

    #error_page 404 /404.html;
    # redirect server error pages to the static page /50x.html
    #

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ /\.php$ {
    #    proxy_pass http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on
    127.0.0.1:9000
    #
}
```

```
#location ~ /\.php$ {
#    root          html;
#    fastcgi_pass   127.0.0.1:9000;
#    fastcgi_index  index.php;
#    fastcgi_param  SCRIPT_FILENAME /scripts$fastcgi_script_name;
#    include        fastcgi_params;
#}
# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
#location ~ /\.ht {
#    deny all;
#}
}
```

Una vez terminado de editar el archivo de configuración default.conf, creamos el contenedor de Nginx con el siguiente comando.

```
sudo docker run --name nginx --restart=always -p 80:80 -p 443:443 \
-v /proyecto/nginx/html:/usr/share/nginx/html \
-v /proyecto/nginx/default.conf:/etc/nginx/conf.d/default.conf \
-v /proyecto/nginx/.htpasswd:/etc/nginx/.htpasswd \
-v /proyecto/nginx/SSL:/etc/nginx/SSL -d nginx
```

B. Programación del dispositivo ESP32

```
esphome:
  name: modulo3
  platform: ESP32
  board: esp32dev

wifi:
  ssid: "[REDACTED]"
  password: "[REDACTED]"
  # Optional manual IP
  manual_ip:
    static_ip: 192.168.1.68
    gateway: 192.168.1.1
    subnet: 255.255.255.0
  # Enable fallback hotspot in case wifi connection fails
  ap:
    ssid: "Modulo3 Fallback Hotspot"
    password: "nDgsHubTqEzc"

captive_portal:

mqtt:
  broker: 192.168.1.60
  username: [REDACTED]
  password: [REDACTED]

# Enable logging
logger:

i2c:
  id: ccs
  sda: GPIO21
  scl: GPIO22
  scan: True

ota:
  password: "[REDACTED]"

dallas:
  - pin: GPIO4

sensor:
  - platform: dallas
    address: 0xF1000008B68E6B28
    name: "Temp2ds18b20"
    accuracy_decimals: 2

  - platform: dht
    pin: GPIO12
    temperature:
      name: "DHT11_Temp"
      accuracy_decimals: 2
    humidity:
      name: "DHT11_Humidity"
    update_interval: 60s
    model: DHT11

  - platform: dht
```

```
pin: GPIO14
temperature:
  name: "DHT22_Temp"
  accuracy_decimals: 2
humidity:
  name: "DHT22_Humidity"
update_interval: 60s
model: DHT22

- platform: ccs811
  i2c_id: ccs
  eco2:
    name: "CCS811 eCO2"
  tvoc:
    name: "CCS811 COVs"
  update_interval: 60s

- platform: adc
  pin: GPIO34
  name: "MQ7_CO"
  update_interval: 150s
  unit_of_measurement: "v"

- platform: adc
  pin: GPIO35
  name: "MQ138_Formaldehido"
  update_interval: 60s
  unit_of_measurement: "v"
```